Project No. 11-3066

Developing the User Experience for a Next Generation Nuclear Fuel Cycle Simulator (NGFCS)

Fuel Cycle Research and Development

Paul P.H. Wilson
University of Wisconsin-Madison

In collaboration with:
University of Idaho
University of Texas-Austin
University of Utah

B.P. Singh, Federal POC
Brent Dixon, Technical POC

NEUP Nuclear Energy University Programs
U.S. Department of Energy

Final Technical Report for NEUP Project 00120341

Developing the User Experience for a Next Generation Nuclear Fuel Cycle Simulator (NGFCS)

10/4/2011 - 9/30/2015

Principal Investigator: Paul P.H. Wilson
University of Wisconsin-Madison

Collaboratoring Investigators:
Anthony Scopatz, Dominique Brossard & Dietram Scheufele, University of Wisconsin-Madison
Erich Schneider, University of Texas-Austin
Valerio Pascucci & Yarden Livnat, University of Utah
Robert Hiromoto, University of Idaho

Other Contributors:
University of Wisconsin-Madison
Kathryn Huff, PhD Student
Matthew Gidden, PhD Student
Robert Carlsen, PhD Student
Arrielle Opotowsky, PhD Student
Zach Welch, MS Student
Ashley Anderson, PhD Student
Nan Li, PhD Student
Meghan McGarry, Post-doctoral Associate

University of Texas-Austin
Robert Flanagan, PhD Student

Executive Summary

This project made substantial progress on its original aim for providing a modern user experience for nuclear fuel cycle analysis while also creating a robust and functional next-generation fuel cycle simulator.

The Cyclus kernel experienced a dramatic clarification of its interfaces and data model, becoming a full-fledged agent-based framework, with strong support for third party developers of novel archetypes. The most important contribution of this project to the the development of Cyclus was the introduction of tools to facilitate archetype development. These include automated code generation of routine archetype components, metadata annotations to provide reflection and rich description of each data member's purpose, and mechanisms for input validation and output of complex data.

A comprehensive social science investigation of decision makers' interests in nuclear fuel cycles, and specifically their interests in nuclear fuel cycle simulators (NFCSs) as tools for understanding nuclear fuel cycle options, was conducted. This included document review and analysis, stakeholder interviews, and a survey of decision makers. This information was used to study the role of visualization formats and features in communicating information about nuclear fuel cycles.

A flexible and user-friendly tool was developed for building Cyclus analysis models, featuring a drag-and-drop interface and automatic input form generation for novel archetypes. Cycic allows users to design fuel cycles from arbitrary collections of facilities for the first time, with mechanisms that contribute to consistency within that fuel cycle. Interacting with some of the metadata capabilities introduced in the above-mentioned tools to support archetype development, Cycic also automates the generation of user input forms for novel archetypes with little to no special knowledge required by the archetype developers.

Translation of the fundamental metrics of Cyclus into more interesting quantities is accomplished in the Cymetric python package. This package is specifically designed to support the introduction of new metrics by building upon existing metrics. This concept allows for multiple dependencies and encourages building complex metrics out of incremental transformations to those prior metrics. New archetype developers can contribute their own archetype-specific metric using the same capability. A simple demonstration of this capability focused on generating time-dependent cash flows for reactor deployment that could then be analyzed in different ways.

Cyclist, a dedicated application for exploration of Cyclus results, was developed. It's primary capabilities at this stage are best-suited to experienced fuel cycle analysts, but it provides a basic platform for simpler visualizations for other audiences. An important part of its interface is the ability to fluidly examine different slices of what is fundamentally a five-dimensional sparse data set. A drag-and-drop interface simplifies the process of selecting which data is displayed in the plot as well as which dimensions are being used for grouping/aggregation and filtering. A purpose-built visualization was created to display flows between different groupings of agents and interrogate those flows over time.

A remote, cloud-based execution capability, known as Cloudlus, was developed and deployed for multiple purposes. A primary motivation of was to enable client-server operation, thus making Cyclus available on a wider variety of platforms. Since the Cyclist user interface was written in Java, it could run on any desktop platform and use a network connection to

run the simulation on a cloud server. This facilitated distribution to a wider audience and prototyped the technology necessary for support of mobile/tablet operation. This capability also proved to be valuable for large scale automated analysis, in which a master process submitted work to hundreds of Cloudlus execution nodes for optimization work.

Although the various thrusts of this project never achieved as tight integration as was originally intended, important progress was made - and insights gained - in each thrust. Changes in project scope required refocusing on fewer stakeholder groups and undermined the ability to develop an improved user experience for a wide variety of audiences. This is still an interesting extension of this work that can possibly be pursued in the future.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Interest in NFCSs by the U.S. Department of Energy (USDOE) Office of Nuclear Energy intensified in the mid-2000's as a variety of policy issues converged to prompt a renewed interest in advanced nuclear fuel cycles. Whereas closed fuel cycles have always been imagined as a way to extend uranium resources, their ability to reduce the burden of radioactive waste management was a new motivation, as was the possibility of reducing global proliferation risk by implementing international nuclear fuel cycle services. With a richer set of possible benefits, new tools were necessary to provide insight into the degree to which different technology options would have an impact on these and other socio-political metrics.

The fundamental purpose of a NFCS is to track the mass flows and isotopic composition of material as it moves among facilities in a complete nuclear fuel cycle. At a minimum, the set of facilities should include nuclear reactors for the consumption of uranium and the production of transuranics and fission products, but most tools include front-end facilities including mines, enrichment facilities and fuel fabrication facilities, as well as back-end facilities including storage and separations. As they become more sophisticated, a NFCS will also track information about facility deployment, utilization and decommissioning, in order to support economic assessments that are dominated by the capital costs and capacity factors of those facilities.

A variety of such tools emerged, often developed to answer specific programmatic questions. Most notable among these tools were Dymond[15], DANESS[16] and VISION[17] all initially developed at national laboratories and sharing a common heritage. Although they continue to provide value to the nuclear fuel cycle analysis community, a number of limitations were identified in these tools, leading to a desire for a next-generation fuel cycle simulator (NGFCS).

One limitation of the previous tools was their reliance on a systems dynamics framework. Such frameworks were initially considered attractive because of the graphical development environments of stocks and flows that mapped well onto the material flow modeling at the core of a NFCS. However, the stock and flow paradigm requires all models to be expressed in the context of a first-order ODE solved with an Eulerian approach. Many models are not well-suited to this approach and require the implementation of alternative methods as extensions to the systems dynamics framework using a more traditional programming interface, thus diminishing the benefit of the graphical development environment. Systems dynamics platforms also suffered from limitations on the size of problem they could accommodate,

particularly early in the development of these tools. As it became necessary or desirable to expand the number of isotopes tracked in material flows and incorporate large matrices to represent the long term response of those material flows, limits in the total data footprint were reached. NFCSs were often the largest models to ever be implemented in these platforms and close collaboration with their developers led to improvements that relaxed these limits. The systems dynamics platforms were not well-suited to collaborative development because contributions from individual developers could only be merged manually. Finally, these platforms required the purchase of commercial licenses for each user at each site, both costly and an obstacle to deployment at large scales for parametric studies, sensitivity analysis and/or optimization.

Another limitation arose from the problem-focused development path for these tools. Because it was necessary to prioritize particular candidate fuel cycles, they each developed depth in their modeling fidelity to support those specific fuel cycles and without sufficient modularity to quickly incorporate novel technologies. As policy-driven interest in specific fuel cycles waned, demand grew for these tools to analyze the fuel cycle impact of a variety of novel nuclear energy technologies. Incorporating these technologies in the existing tools required a steep learning curve to first learn the particular systems dynamics platform and then understand the modeling approaches used to incorporate the complexity of real fuel cycles. This was often beyond the scope of many small technology development efforts, especially when crude estimates of the steady-state fuel cycle impacts were possible with simple spreadsheet models. At the same, time each of the tools had implemented slightly different assumptions and approximations rendering comparison a challenge and ultimately undermining confidence in their ability to produce reliable results.

Finally, the user experience for these tools was never a focus during their development, despite the recognized value in enabling their use by a broader set of audiences to improve the communication of their results. The graphical nature of the systems dynamics platforms was considered to be sufficient until the implemented models became increasingly complex. Fuel cycle scenarios were defined by data in ever-growing spreadsheets and results stored in even bigger spreadsheets. Visualization of both input and output data was generally limited to the spreadsheet software's capabilities.

Therefore, the Fuel Cycle Research & Development program chose to support the development of a NGFCS platform with sufficient modularity to accommodate a wide variety of audiences, use cases and developer needs. The requirements for this framework included the modeling of discrete transactions of materials among a set of discrete facilities operating in distinct geographic regions. All transactions would arise as the result of resolving a set of offers and requests in a market whose transaction costs may be determined by a wide array of economic, political and technical parameters. Most importantly, a modular plug-in architecture would allow user-developers to introduce different models with alternative market and facility behaviors, thereby isolating variations in modeling to single changes. The fundamental data generated by such a framework includes a list of material transactions and the facilities involved in such transactions, a history of the isotopic composition of each material object, and an operating history for each facility. As additional plug-in modules are developed, additional data will be generated related to the facility histories, material histories, or market transactions.

The NGFCS was also expected to be a useful evaluation tool for a variety of audiences.

Non-technical audiences would interact with the NGFCS in a way that allowed them to express critical high-level decisions and understand the outcomes of those decisions in a set of key metrics. At the same time, expert audiences might be interested in a quantitative technology assessment to help motivate design improvements. Developers of the NGFCS would need to visualize their results to ensure consistency and correctness. Given the wide array of possible metrics and summaries, each type of user might want to gradually explore the results with increasing depth and sophistication. At the same time, it was important that the NGFCS allow experts to introduce new modules to capture specific physical models or market behavior, but rely on a common infrastructure that facilitates direct comparisons of results.

1.1 Research Plan Overview

This project was therefore designed to explore strategies and develop tools to enhance the user experience for the NGFCS for a range of target audiences. The research plan identified five thrusts:

1. Stakeholder, parameter and metric identification
2. User interface and model generation
3. Metric translation
4. Visualization environment
5. Efficient design of a client-server model

The work scope definition for this project referred to a NGFCS platform that would be developed by the USDOE and requested proposals for enhancements to that platform, explicitly prohibiting proposals that aimed to develop the platform itself. However, at the time that this project was awarded, no such platform existed and there were no plans to create one. Given ongoing efforts to design and implement such a platform at the lead institution, a renegotiated scope and research plan was developed, inserting an additional Thrust 0 that would dedicate resources to the completion of the Cyclus framework as the NGFCS. The addition of this thrust required some scaling back of the scope of each of the other thrusts with the ultimate research plan outlined below.

Thrust 0 was lead by nuclear engineering software developers to finalize and implement the design of Cyclus. While most capabilities had been demonstrated in some fashion, this project allowed for robustness and stability improvements that enable a broader adoption and simplifying the process of extending the platform with new models.

Thrust 1 was led by experts in communication science, and specifically in the science of communicating about controversial science. This critical project element was designed to enhance the relevance of the NGFCS to a wider audience than the traditional nuclear engineering audience of previous tools by identifying the metrics that address the questions and concerns of those audiences rather than focusing on the metrics that the nuclear engineering community values. This thrust incorporated a thorough review of prior testimony and historical documents to identify stakeholders and the issues that they raised, interviews with a variety of individuals representing those stakeholder groups to further identify the important issues, a survey of a larger community of stakeholder to develop a quantitative understanding of how those issues were related to each other, and an experiment to understand how different

forms of visualization would communicate the results of a fuel cycle simulation. A detailed account of the work performed under this thrust is found in Chapter 3.

The primary purpose of Thrust 2 was to develop a convenient graphical interface by which users could define the fuel cycle scenarios of interest to them. Inspired by the drag-and-drop promise of the systems dynamics paradigm, this interface would allow users to build high-level graphical representation of a nuclear fuel cycle by placing and connecting individual facilities. Particular consideration would be given to the rules for connecting facilities to reduce the depth of nuclear engineering domain knowledge expected of users while still preserving a physically valid flow path for material. The interface would expose varying level of detail, with many default values hidden from users seeking a simpler experience, but available to those users looking to control all parameters. Each of these parameters would be self-documented by the interface allowing users to quickly learn what the valid purpose and range of each parameter is. Chapter 4 outlines the development at testing of such an interface.

Thrust 3 focused on translating the fundamental data recorded by the NGFCS into metrics identified by Thrust 1 as interesting to stakeholders. Since the NGFCS would operate at the fidelity of discrete material objects, perhaps individual fuel assemblies, being transacted by discrete facilities, a minimum amount of data would be recorded directly in its fundamental output. This data might include, for example, the masses and compositions of individual material objects being transacted. Although some applications may be interested in such high-fidelity data, many applications would be integrated in integrated quantities such as total flow rates and or inventories in the fuel cycle. Moreover, there may be interest in quantities derived from this aggregate data such as decay heats or radiotoxicity. Finally, given the role of modularity and extensibility in the NGFCS itself, it is important for this metric translation capability to offer similar extensibility. Chapter 5 describes some sample metrics and the conceptual framework embodied in an extensible metric translation capability.

The ability for user to explore their data in an organic and discovery-driven manner was the goal of Thrust 4. Beyond simply providing simple plots to represent the output metrics (identified in Thrust 1 and calculated in Thrust 3), the user may want to aggregate some results over select dimensions and filter those results across other dimensions. When those features are shared over a set of related visualizations, they form a workspace in which all the results can be manipulated as a set. Chapter 6 describes the implementation of these concepts as well as a custom visualization for the flow of material among different fuel cycle components.

Although it was not clear how much computational power would be necessary to run a single case with the NGFCS, it was likely that certain use cases would require remote operation in order to provide sufficient computing capability, particularly if thin clients such as tablets are imagined as ultimate user interface devices. Thrust 5 considered the factors that would facilitate a client-server mode of operation in which the simulation itself was operating remotely and only the visualization tools were local to the PC, tablet, or smartphone. This resulted in the development of a cloud-like simulation capability that can support remote execution of single simulations submitted via a website or with the graphical interface developed in Thrust 2, as well as large scale analysis by deployment as a compute farm.

Chapter 2

Thrust 0: Cyclus Infrastructure Development

Under Thrust 0, the Cyclus framework was advanced from a conceptual design to a fully functional software tool. As this project began, the fundamental features of Cyclus were in place, but it was far from a usable tool for fuel cycle simulation. Through the efforts of this project, Cyclus emerged as a more usable and stable software platform for nuclear fuel cycle analysis. The overarching goal was to provide maximum flexibility so that changes in fuel cycle design did not require substantial retooling of the software. Specific design features that support this flexibility include:

1. **Modular design** to support extensibility with runtime swappable facility models.
2. **Agent-based approach** to incorporate social/behavior interaction models to govern how each facility interacts with others.
3. **Individual facility modeling** to allow each facility to operate in a distinct manner, allowing modeling of startup, shutdown and other disruptions.
4. **Discrete material tracking at the nuclide level** to capture the effects of individual facility performance and enable forensic tracking of material object ownership.
5. **Minimal physics assumptions** to enable different users to invoke low fidelity, systems level models or high fidelity, facility level models, or anything in between.

Under this project, the Cyclus effort developed into an *ecosystem* composed of:

- an open source simulation kernel, known as Cyclus,
- many plug-in modules, each implementing a particular model for a facility, institution or region, and known as *archetypes*, and
- an open source analysis and visualization tool, known as Cyclist.

The first section of this chapter will describe the fundamental design and implementation of the Cyclus kernel while the following section will describe the infrastructure to support the development of new archetypes.

2.1 Fundamental Design and Implementation

More details regarding the implementation and demonstration of the fundamental design of the Cyclus kernel are available in ref [1].

2.1.1 Modularity

The modular nature of Cyclus is a key element to its ability to remain flexible, and also drives many subsequent design decisions. The main goal of this flexibility is to allow for the computational models that represent individual agents (see Section 2.1.2) to be swapped easily, as a user option. Two specific advantages of this choice are that (a) individual researchers can focus their efforts on new modules that match their expertise and support their needs, and (b) the impact of changing modeling choices can be explored in a consistent environment by swapping only one module at a time.

In its purest expression, each module operates in a so-called *black-box* fashion. That is, any single module has no knowledge of the internal state or behavior of any other module and can only interact in a narrowly defined and specific way. In addition to imposing a number of software engineering limitations on the implementation of the kernel and archetypes, this design goal also requires careful design of that interaction interface among the modules.

Figure 2.1 illustrates this concept, indicating the different API for each type of agent, and the possibility of different implementations in each case. Different implementations of each type of facility, for example, may use different models to describe the physics and/or the interaction behavior of that facility. These differences can represent small perturbations from other implementations or entirely different approaches, including wrapping other software. Because the modules can be swapped at runtime, without requiring the software to be recompiled, this also facilitates different distribution modes for these modules, as shown in Figure 2.2, including fully open source, export controlled, and/or proprietary commercially licensed modules.

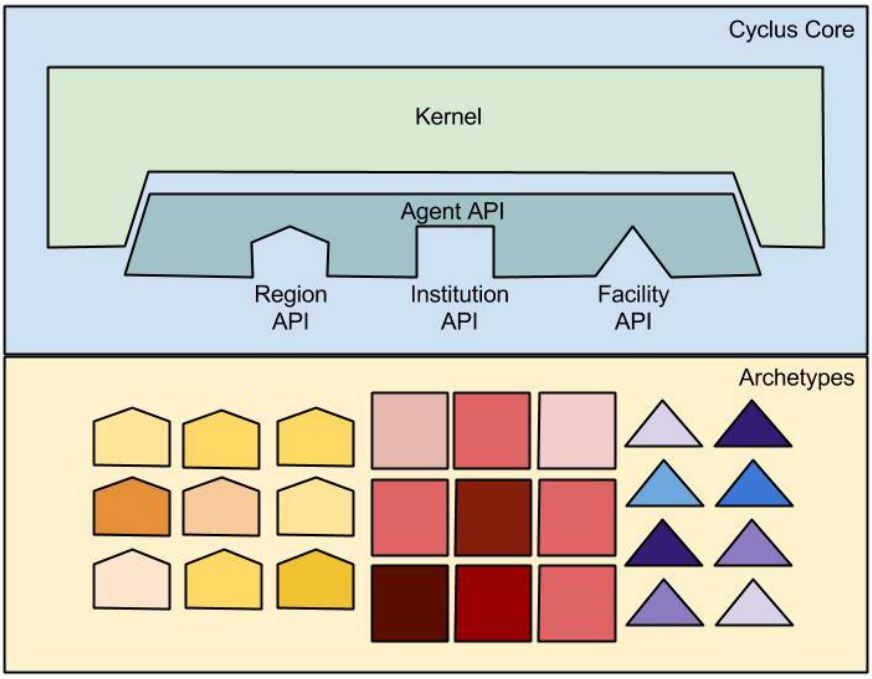


Figure 2.1: The Cyclus core provides APIs that abstract away the details in the kernel and allow the archetypes to be loaded into the simulation in a modular fashion. [1]

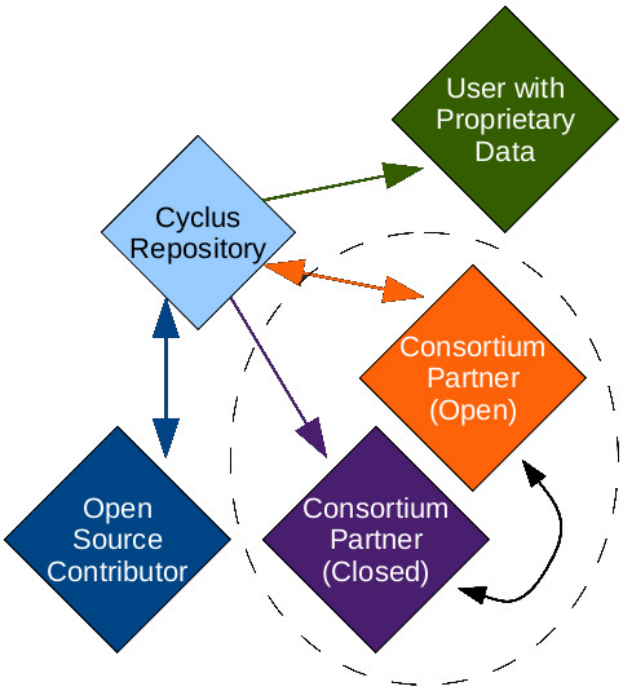


Figure 2.2: The CYCLUS framework enables fully open, partially open, and fully closed collaborations[2].

2.1.2 Agent-based Paradigm

CYCLUS implements an agent-based modeling (ABM) paradigm. In addition to being a natural consequence of the modularity described above, the ABM paradigm is more flexible and intuitive (from both a developer and user perspective) than the system dynamic approach used in current simulators. System dynamics is a popular approach for modeling nuclear fuel cycles [17, 16, 18, 19]. Formally however, system dynamics models are simply a strict subset of agent-based models [20]. That is, any system dynamics model can be translated into an agent-based model. ABM techniques therefore enable a broader range of simulations in a more generic fashion.

To ensure interchageability, all agents are based on a common set of object-oriented APIs defined as part of the CYCLUS kernel that define how they interact with other agents and with the environment. These APIs standardize and simplify the implementation of required agent behavior such as trading resources. Optional *mixins* are also provided in a toolkit to provide community standards for some optional behaviors. More details on archetype development are in Section 2.2.

CYCLUS provides a novel representation of entities in the nuclear fuel cycle that reflects the reality in international nuclear power: *facilities* implementing individual fuel cycle technologies, *institutions* managing those facilities, and *regions* providing geographical and political context for institutions and facilities. Under this paradigm, regions and institutions are first class agents much like facilities: they can be developed as stand-alone modules to implement different behaviors and swapped at runtime. CYCLUS implements a Region-Institution-Facility (RIF) relationship through a parent-child hierarchy where regions are

the parents of institutions which are, in turn, the parents of facilities. In this structure, institutions are typically responsible for making decisions to deploy and decommissioning facilities, while regions are typically responsible for establishing an energy demand. Both regions and institutions can have an impact on inter-facility resource flow during the resource exchange process.

2.1.3 Discrete Objects

Implied by this ABM paradigm is that Cyclus models facilities, institutions, regions, and resources as discrete objects, allowing for a range of modeling granularity. This is in contrast to the fleet-based approach used in most other fuel cycle simulators, although it is possible to implement Cyclus archetypes that represent fleets of identically behaving facilities[3]. This discrete facility capability is necessary to enable the tracking of discrete material objects. Examples of use cases supported by discrete material tracking include spent fuel storage, transport and disposal models that require representation of discrete casks and the individual isotopic compositions of their contents.

Another such use case seeks to capture system vulnerability to material diversion. Provenance and trade-history of distinct materials is the fundamental information unit in such studies, and so this type of analysis requires discrete simulation of a target facility and the individual materials modified within it. Material risk analysis, therefore, demands that both facilities and materials should be discretely modeled objects like those in Cyclus.

Cyclus represents these discrete tradable objects generically as *resources*, with a specific implementation for *materials* to represent typical nuclear materials with a mass and nuclide composition. Typical operations on material objects, e.g. splitting, combining, decay, *etc.*, are tracked in detail to support simulation wide mass conservation. The history of ownership and transfer of each object is also tracked.

Material objects in Cyclus can have compositions with an arbitrary number of nuclides and can be optionally subject to radioactive decay, either at the discretion of the facilities which own/manage the material, or whenever it is observed/traded in the simulation. In large simulations, this tracking of discrete material objects can lead to many objects that change frequently. A number of implementation details exist to minimize the impact on system memory and output database file size.

A critical capability at the intersection of the swappable archetype and the discrete resource object paradigms is the dynamic resource exchange (DRE) which implements a market mechanism at each time step for matching consumer agents' requests for material with supplier agents' offers for material. The DRE is described in detail in Ref [21]. Supporting the general Cyclus philosophy, facilities are treated as black boxes and a supply-demand communication framework is defined. The primary development of the DRE was supported under a different project and is reported there[22].

The DRE is a novel simulation concept in the nuclear fuel cycle domain. It provides a flexible supply-demand infrastructure, supporting dynamic flows of resources between agents, even as those agents enter and leave the simulation, and even when those agents are defined by archetypes of arbitrary complexity. Trading between agents can be affected by both the proposed quality of a resource and agent relationships through the use of preferences. Accordingly, a wide range of possible effects can be modeled, from capacity-limited fuel supply

to international trade agreements.

2.1.4 Simulation Support

So that users and developers can build working simulations in the shortest time possible, the Cyclus ecosystem provides fundamental building blocks: basic archetypes and a toolkit of commonly needed functions. The Cycamore library provides a suite of fundamental Region, Institution, and Facility archetypes, while the Cyclus toolkit provides assistance to developers.

Cycamore [4], the Cyclus additional module repository, provides a fundamental set of dynamically loadable libraries providing agent archetypes for basic simulation functionality within Cyclus. Since Cyclus relies on external archetypes to represent the agents within a simulation, Cycamore provides the basic archetypes a new user needs to get started running simple simulations. These archetypes support a minimal set of fuel cycle simulation goals and provide, by example, a guide to new developers who would seek to contribute their own archetypes outside of Cycamore.

As of version 1.5, Cycamore contains one region archetype, two institution archetypes, and eight facility archetypes. Short descriptions of these functions can be found in Table 2.1.

In addition to the core functionality of the Cyclus kernel, which is focused on the set of capabilities needed to implement an agent-based simulation with DRE, a toolkit is provided to assist developers and users with robust and standardized implementations of related simulation and nuclear engineering tasks. The toolkit is an actively developed part of Cyclus, with a primarily forward-looking focus on supporting interesting *in situ* metric analysis tools. The toolkit contains methods to support demand-constrained facility agent deployment with methods that describe different shaped curves for electricity demand, and then assess the available capacity relative to those demand curves, deploying new facilities when necessary. The toolkit also includes nuclear engineering specific methods, such as models to calculate the relationship between material quantity, quality and enrichment capacity.

2.1.5 Quality Assurance

To garner the trust of a broad user and archetype developer community, the Cyclus project must implement a strategy to assure the ongoing quality of the software it provides. Multiple strategies, collectively known as *quality assurance (QA)*, have been developed by the scientific software development community to mitigate structural and algorithmic errors in software. Cyclus has adopted the best practices of the open source software community for ensuring software quality including:

- version control: careful control and cataloging of changes to the software with full attribution of each change to each author,
- code review: every individual change is reviewed by another developer for many aspects, from style to algorithmic performance to correctness, and
- continuous integration: automated testing of individual software units and integrated solutions as each change is proposed and incorporated.

Ref [1] provides more details on each of these topics.

Table 2.1: The Archetypes in CYCAMORE seek to cover a large range of simple simulation use cases[4].

Type	Archetype Name	Functionality
Facility	EnrichmentFacility	This facility enriches uranium at a specified capacity.
Facility	FuelFab	This facility fabricates fuel material based on separated streams.
Facility	Reactor	A reactor model that handles batch refueling, based on pre-determined recipes of compositions. It requests any number of input fuel types and transmutes them to static compositions upon discharge.
Facility	Separations	This facility separates an incoming material into specified streams.
Facility	Storage	This facility holds incoming material for a specified time until before it is offered.
Facility	Mixer	This facility combined a set of incoming streams into a single stream.
Facility	Sink	This facility is capable of accepting a finite or infinite quantity of some commodity produced in the simulation.
Facility	Source	This facility generates material of the composition and commodity type specified as input.
Institution	ManagerInst	The manager institution manages production of commodities among its facilities by building new ones as needed.
Institution	DeployInst	This institution deploys specific facilities as defined explicitly in the input file.
Region	GrowthRegion	This region determines whether there is a need to meet a certain capacity (as defined via input) at each time step.

2.2 Facilitating Archetype Developers

A principal distinguishing feature of CYCLUS is the ability to introduce novel agents by developing customized archetypes. Not only is this a departure from the structure of most NFCS tools, but the degree of specialization that is necessary to represent different fuel cycle facilities is a departure from many agent-based simulation frameworks. Each agent must interact with the CYCLUS kernel and other agents using a rich interface that controls initialization of agent parameters, agent interaction with the DRE, initiation of agent behavior between trades, decommissioning of agents, and recording of all agent data. Correct implementation of these interactions with the kernel are necessary, in addition to implementing the physics and interaction behaviors that are the true motivation for the new agent archetype.

To mitigate the difficulties of writing archetypes, CYCLUS aids in overcoming the hurdle of interfacing with the kernel itself. Additionally, to the extent possible, CYCLUS should also provide tools that ease the implementation of the physics and desired behavior. A variety of

strategies are used by CYCLUS to ease archetype development:

- **Automatic Model Templating:** By automatically inspecting and creating portions of archetypes, the overhead of adhering to the CYCLUS API is removed. This is performed via preprocessing archetypes and then automatically generating the necessary C++ source code to implement many parts of the necessary interface.
- **Data Communication Protocol:** CYCLUS provides a common basis for archetypes to store and retrieve complex data in its database. This is achieved through the implementation of CYCLUS-specific type system. This alleviates the need for archetype developers to invent and implement custom solutions for storing the data that is generated by the archetype.
- **Metadata Annotations:** Archetypes have a standard place to store and retrieve both pre-defined and arbitrary metadata. This allows for archetype developers to communicate relevant information to tools outside of CYCLUS (such as a visualization tool).
- **Input Validation:** CYCLUS uses an Extensible Markup Language (XML) input file format because it allows for a the definition of a schema that can be used to automatically validate the input file. The schema for each archetype is automatically generated by the CYCLUS tools, relieving the developer of this burden while still allowing them to impose automatic validation.
- **Model Location:** CYCLUS has a packaging system for archetypes and libraries of archetypes. This provides a standard mechanism for searching for and locating models on a user's system. Furthermore, all archetype developers can uniquely specify their own archetypes without the fear of overlapping names. For example, two developers could each have a `Reactor` archetype, but they would exist in different packages and be disambiguated.

Though the above mechanisms are discussed with respect to CYCLUS, they are transferable to any agent-based modeling framework that requires modular agent archetypes. The following subsections present greater detail regarding these strategies. Further detail and examples are available in Ref. [5].

2.2.1 Automatic Model Templating

Every CYCLUS archetype is required to implement a standard set of member functions that control the definition of a prototype that is derived from that archetype (`InfileToDb`), the life cycle of each instance of that archetype (`InitFrom`,`Clone`,`InitInv`), and the interaction of the agent with the database that stores the state of each agent (`Snapshot`,`SnapshotInv`). There are additional option functions that allow an archetype to take specific action when it is deployed (`Build`, `EnterNotify`), when it chooses to deploy or decommission its own children (`BuildNotify`,`DecomNotify`), or when it is decommissioned itself (`Decommission`).

Since each archetype is free to define a unique and independent set of member data and behaviors, these methods must be specific to the archetype so that they can be sure to address all the correct data members. Manually generating all of these functions is tedious, time consuming, error prone, and may require specialized knowledge of the CYCLUS kernel that would be otherwise unnecessary. It is, however, possible to automatically generate the code that implements the above functions based on the set of members that are defined in

the archetype. Furthermore, with a small amount of additional metadata (see Section 2.2.3 below) for each member variable, it is also possible to automatically generate the correct XML schema for validation of input to that archetype.

A custom pre-processor, called `cycpp`, has been written to read and parse the simplified definition of an archetype and automatically generate the correct methods to implement the above interface. The pre-processor replies on metadata that is attributed to the member data through the use of `#pragma cyclus` directives. These are otherwise skipped by the compiler, but can be used to identify special behavior.

2.2.2 Data Communication Protocol

Cyclus relies on a number of alternative storage formats for the data it generates, allowing the user to prefer either typical Structured Query Language (SQL) databases or numerically oriented Hierarchical Data Format version 5 (HDF5) files. Both of these systems have full support for simple data types, but much of the data generated in a fuel cycle simulation has more complex relationships. The Cyclus kernel has implemented an extensible system to support complex types. This is important for archetype designers/developers because it allows them to use a richer and more expressive set of data structures, and reduces possible errors that might be introduced by having to transcribe those data structures in/out of simpler forms for storage. This is also an important feature to enable automatic model templating: `cycpp` is able to determine the "shape" of complex data structures and invoke the correct tools for storage of that data.

2.2.3 Metadata Annotations

As mentioned above, automatic code generation relies on metadata that is either determined automatically from the archetype declaration or is introduced in optional `#pragma cyclus` directives.

One use of this metadata is to define the behavior of archetype member data, and provide a high-level mechanism to invoke many low-level behaviors through the `cycpp` pre-processor. For example, a simple declaration of the shape of a data type can be used to generate the code that will invoke the correct data storage behavior in various formats.

All metadata, whether required or optional, is stored as part of the archetype and available during a Cyclus simulation for general use. At a fundamental level, this allows for reflection in a way that is not typical of C++ software: the archetypes are aware of their own name and the types of their member variables.

Finally, the annotations can be particularly valuable beyond the Cyclus kernel. Cyclus provides a mechanism to query those annotations for use in other tools that support input file generation and output visualization. Several standard annotations are available to support input file generation, including declaring a default value, different forms of documentation, and physical units. Use of this data will be discussed further in Chapter 5.

2.2.4 Input Validation

A Cyclus input file consists largely of the definition of agent *prototype*, each formed by configuring an archetype with a set of parameters. In many software tools that rely on user input, validating that the users choices for those parameters makes sense in some way can be a substantial task. This kind of validation includes checking whether certain values are within physical bounds, whether all values are defined with the correct type, and whether there are enough entries for complex data types. By relying on XML as the input file format, Cyclus is able to invoke its built-in grammar-based validation capability. A *schema* provides a definition of the correct grammar against which the input file is tested.

In most cases, the schema for an archetype can be automatically generated from the archetype declaration, with some additional information collected from annotations. This allows archetype developers to have the full advantage of a grammar-based validation without having to learn the details of how those grammars are defined. Grammar for the validation of data type and structure can be derived by inspecting the type and structure of the member variables themselves. Grammar that validates against specific semantic meanings (e.g. a variable must be positive) requires metadata annotations. The metadata annotations also provide a mechanism for developers to completely override the auto-generation of the schema.

2.2.5 Model Location

The final capability provided to archetype developers is a standardized way to declare the location of a specific archetype. Since archetypes are dynamically loadable, it is necessary for the Cyclus kernel to find each archetype at runtime. The simplest strategies for finding archetypes rely on them all being named in a standard way in a standard location in the file system. This is both constraining and presumes a level of coordination among archetype developers that undermines the flexibility of the plugin archetypes. Instead, an archetype specification mechanism has been developed to allow both flexibility and disambiguation of archetypes at runtime. The specification allows a user to define where in the file system a plugin library is located, the name of that library, and the name of the archetype within that library. This means that different developers can choose the same name for their library of archetypes, as long as it is stored in a different location. Similarly, they can have similar names for their individual archetypes without worrying about name collision.

This set of tools and capabilities not only lowers the barrier to the development of new archetypes, allowing developers to focus on the specific physical and behavior models of interest to them. The software development tasks of agent instantiation, data persistence, and input validation are handled automatically. This allows the Cyclus ecosystem to use advanced technologies for these purposes without raising the cognitive burden on archetype developers to learn all of those technologies.

2.3 Summary

This project enabled the Cyclus effort to develop into a robust tool for fuel cycle simulation, meeting its primary design goals while also facilitating contributions from a wide set of archetype developers. This product of Thrust 0 was a next-generation fuel cycle simulation

framework suitable for the user experience extensions studied in the other thrusts of this project.

Chapter 3

Thrust 1: Stakeholder Engagement

One of the aims of this project was to make fuel cycle simulation and analysis accessible to a broader audience of users by designing different user experiences for different categories of users. Doing so requires that the interests and use cases of those different categories be sufficiently understood. The overarching goal of Thrust 1 was to determine the specific kind of information that various users and stakeholders would like to learn from a nuclear fuel cycle simulator to support their application of such a tool. This thrust was born from a recognition that different categories of users would be interested in different kinds of information, possibly different from the set of information traditionally provided by the fuel cycle experts who typically study such systems. Moreover, the fidelity and nuance surrounding that information may vary across groups of users.

One of the innovative aspects of this project was the incorporation of social science research as a full partner. This thrust was led by Prof. Dominique Brossard of the Life Science Communication Department at the University of Wisconsin-Madison. This thrust incorporated established methods of social science research to study both the kind of information that users would be interested in learning, as well as the impact of different visualization modes on their understanding of that information. More details about the studies described in this chapter can be found in the PhD dissertation of Dr. Nan Li [6].

The first task in this thrust was to identify different communities of stakeholders and the issues that are most important to those communities. Informed, in part, by this information, the second task assessed how different aspects of a data visualization affected the integration of that data by different stakeholder audiences.

3.1 Stakeholder and Issue Identification

This task followed a well-established social science research protocol for identifying and characterizing stakeholder groups and their most important issues, beginning with a thorough review of documents related to nuclear fuel cycles, continuing with a round of qualitative research in which a set of stakeholders was interviewed, and concluding with quantitative research in the form of a survey of a larger set of those stakeholders.

Because the NGFCS was not sufficiently advanced at the beginning of the research project to support a wide variety of stakeholders, the scope of this thrust was narrowed to focus

on federal decision makers: Congressional staff, employees of federal agencies, and national think-tanks and non-profits.

3.1.1 Document Review & Analysis

The starting point of this research was to collect and review a wide array of publicly available documents related to public policy and public engagement around the nuclear fuel cycle. These documents came from many sources, with a large fraction coming from public testimony before various panels and committees. The most recent set of documents, for example, came from testimony and contributions to the Blue Ribbon Commission on America's Nuclear Future, that had concluded its review just as this project began.

This kind of review is a critical step for establishing first a fundamental knowledge of the vocabulary and terminology, and then using that to identify the different categories of stakeholders. For each category of stakeholder, this analysis also helped to identify the important issues when they consider existing or potential future nuclear fuel cycles. The primary outcome of this phase was the development of an list of potential interviewees and a script to conduct those interviews.

3.1.2 Stakeholder Interviews

Interviews were were conducted with 12 individuals from the nuclear energy policy community, with two purposes. Individuals were identified by reviewing all the public hearing records found by searching for "nuclear" in the ProQuest Congressional Publications database between May 2009 and March 2012. From these documents spanning topics from waste management to profliferation to safety, a cohort of policy experts with varying professional backgrounds, experiences and interests was recruited.

First, the content of these interviews would provide a more nuanced understanding of the issues identified in the document review in order to formulate survey questions for a larger sample. Figures 3.1 and 3.2 show the issue categories identified through this process for nuclear energy in general and advanced fuel cycles in specific. Interviewees were also solicited for specific recommendations of information that a NFCS might be able to provide to help inform their understanding and decision making around nuclear fuel cycles. These specific recommendations were allocated to seven categories:

1. General,
2. Environmental and Health Safety,
3. Waste Management,
4. Cost and Economic Issues,
5. Resource Utilization,
6. Sustainability, and
7. Non-proliferation.

Each category contained at least seven (Sustainability) and up to 38 (Cost and Economic Issues) specific recommendations. Two categories (Environmental and Health Safety and Cost and Economic Issues) had six recommendations that were mentioned by more than one interviewee.

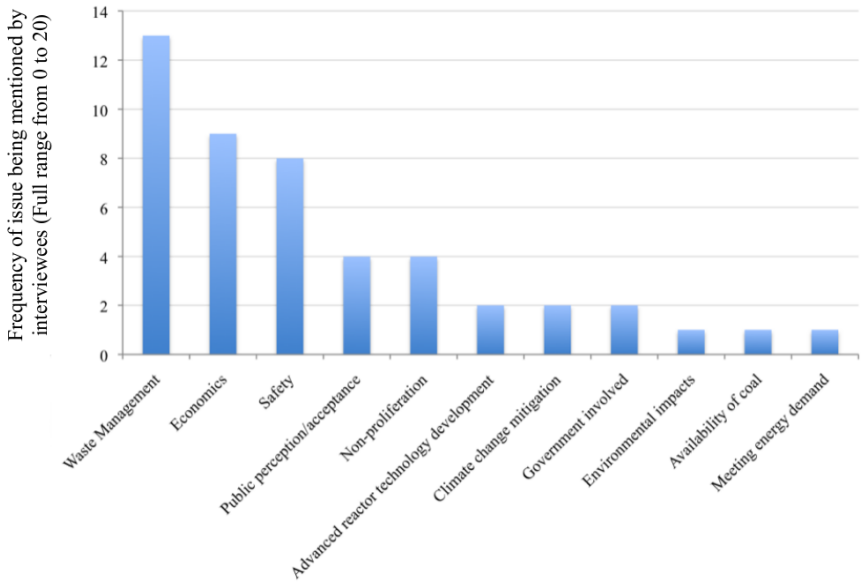


Figure 3.1: Main Issues Related to Nuclear Energy Identified by Qualitative Interviews

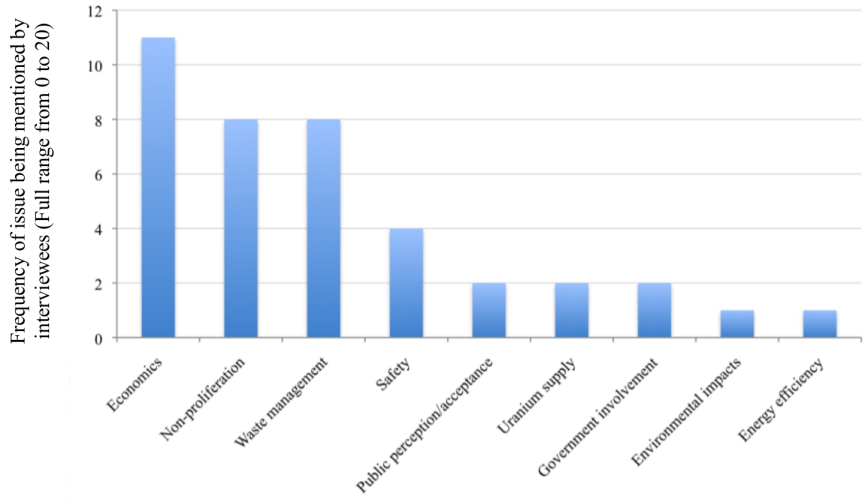
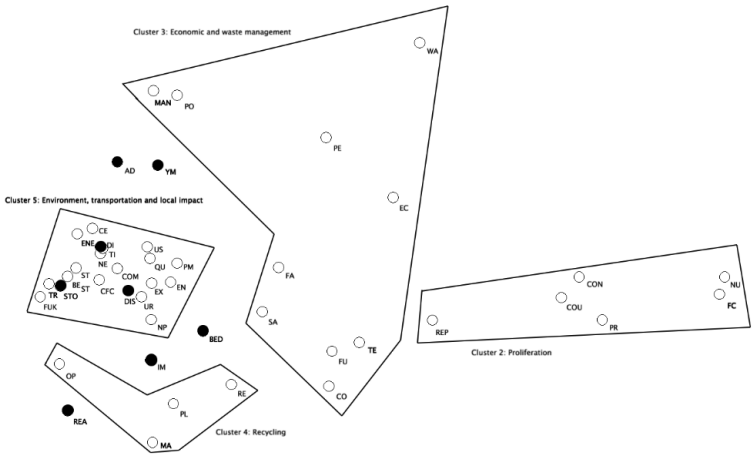


Figure 3.2: Main Issues Related to Advanced Fuel Cycles Identified by Qualitative Interviews

The second purpose of these interviews was to perform semantic computational analysis of the language in the transcripts of those interviews. The CATPAC II software was used to detect patterns and correlations between different conceptual words that appear in the transcripts by recording their frequency of occurring together in a moving 5 word window. The fundamental measures of connection between concepts can be analyzed with a number of techniques, including a multidimensional scaling (MDS) analysis. The MDS analysis results can be visualized in a 2D or 3D space to identify emergent meanings and dominant themes. Figure 3.3 shows such a visualization for the terms identified in the transcripts of the six interviews conducted with government policymakers.



Note: Dark points represent the concepts that do not fit the results of hierarchical cluster analysis

AD. ADMINISTRATION	DI. DIFFERENT	FUK. FUKUSHIMA	PL. PLANT	ST. STATE
BED. BACKEND	DIS.DISPOSAL	IM. IMPORTANT	PO. POLICY	STO. STORAGE
BE. BENEFIT	EC. ECONOMIC	MAN. MANAGEMENT	PM. POLICYMAKER	TE. TECHNOLOGY
CE. CERTAINLY	ENE. ENERGY	MA. MATERIAL	PR. PROLIFERATION	TI. TIME
CFC. CLOSEDFUELCYCLE	EN. ENVIRONMENT	NU. NUCLEAR	QU. QUESTION	TR. TRANSPORTATION
COM. COMMISSION	EX. EXAMPLE	NE. NUCLEARENERGY	REA. REACTOR	US. UNITEDSTATES
CON. CONCERN	FA. FACILITY	NP. NUCLEARPOWER	RE. RECYCLE	UR. URANIUM
CO. COST	FU. FUEL	OP. OPERATING	REP. REPROCESSING	WA. WASTE
COU. COURSE	FC. FUELCYCLE	PE. PEOPLE	SA. SAFETY	YM. YUCCAMOUNTAIN

Figure 3.3: 2D Representation of Unique Concepts for Government Policymaker Interviewees.

A similar analysis for the six interviews conducted with non-profit/think-tank experts showed that different clusters emerge, suggesting nuanced differences in how policymakers and think-tank experts characterize the most important issues facing the development of the nuclear fuel cycle.

3.1.3 Stakeholder Survey

A survey was constructed with seven questions specifically related to nuclear energy, advanced fuel cycles, and NFCSs, followed by three questions that assess the respondents' beliefs on the relationships between science, the media and policy, and five demographic questions (see Appendix B of Ref [6]). Two central questions asked respondents about possible output metrics from a NFCS, specifically asking them to judge both the importance of the metric and the level of confidence with which they believed that metric could be calculated.

The survey was sent to 557 policy stakeholders including:

1. 404 attendees of the public hearings documented above,

2. 63 former members of Congress or their staff that focused on energy issues determined from reviewing the directories of the 109^{th} through 111^{th} Congress, and
3. 90 state and local stakeholders determined by reviewing the public meetings held by the Blue Ribbon Commisison for America's Nuclear Future in 2011.

After a number of rounds of reminders, a total of 137 completed surveys (27.4% response rate) were returned and analyzed.

One type of analysis sought to test a hypothesis identified in the interview analysis that the institutional identity of a respondent would influence their perception of the importance of different issues, as well as their view of the role of science as a driver of policy. This analysis found that industry and advocacy groups were less likely to think that environmental health and safety should be taken into account when developing advanced nuclear fuel cycles, whereas academic stakeholders were less likely to consider nuclear waste management as an important issue for advanced fuel cycle development.

One of the important results of the survey was to sort the issues related to advanced nuclear fuel cycles by their perceived importance as metrics for a NFCS and by the perceived confidence in being able to calculate such metrics. Of the thirteen issues that were ranked with high importance, five of them were also ranked with high confidence while the remaining eight were ranked with low confidence.

- **High Importance, High Confidence**:
  - fuel types and reactor types needed
  - long-term nature of nuclear fuel cycle
  - impact of nuclear fuel cycle on energy generation
  - waste volumes
  - waste types
- **High Importance, Low Confidence**:
  - operating costs
  - costs of entire nuclear fuel cycle
  - cost of waste disposal space
  - amount of sustained funding
  - probability of accidental release
  - impacts on local economies
  - attractiveness to utility companies
  - public acceptance
- **Low Importance, High Confidence**:
  - amount of uranium needed
  - amount of mining
- **Low Importance, Low Confidence**:
  - availability of uranium resource
  - cost of building public support
  - carbon price

The most important issues should be identified as priorities for implementation in a NGFCS and those that are also deemed low confidence should bre identified as research priorities in an effort to reduce their uncertainty and raise confidence in being able to present such metrics.

3.2 Assessing Visualization Efficacy and Credibility

The second major component of Thrust 1 was to assess the effectiveness of different visualization modes for communicating information from a NFCS. Because the CYCLUS simulator was not sufficiently advanced, a surrogate set of visualizations was used for testing. Because both were indicated as important in the prior work, the visualization testing used mock data for costs and waste management. Experiments were constructed in which a respondent was shown a visualization of this data and asked to respond to a set of questions regarding both the data itself and the visualization that was used to communicate the data.

A total of 517 valid responses to the experiment were collected from undergraduate students at the University of Wisconsin-Madison from a variety of disciplines and backgrounds. They were divided roughly evenly between science & engineering (28.7%), social sciences (32.9%), and the humanities (31.9%), and were almost all (98%) between the ages of 18-24. The majority (64%) were females and about the same fraction were either freshman or sophomores.

In each case, the respondents were first asked a series of questions about themselves, designed to probe the following:

- familiarity with nuclear energy issues
- perceived credibility of different information sources
- self-assessment of numeracy and graph literacy

Each respondent was then shown one visualization of mock data related to nuclear fuel cycles. A number of characteristics of the visualizations were varied across the sample of respondents:

- area chart vs bubble chart
- static chart vs interactive chart
- cost vs mass
- government attribution vs academic attribution

Figures 3.4 and 3.5 show two of the sixteen possible combinations.

After viewing one of the sixteen possible variations, respondents were asked a series of questions about the data they had seen. Some questions tested how accurately the respondents could interpret the data, both in absolute terms by asking for specific values, and in relative terms by asking for comparisons. Related questions tested the quality of decision making by asking respondents to identify the best option given particular goals. Other questions tested the opinions of the respondents with regards to the perceived credibility and the perceived effectiveness of the visualization.

Finally, a set of questions was used to further probe the numeracy of the respondent, both in terms of their self reported perceived numeracy and by asking a series of questions about unrelated data.

3.2.1 Graph Comprehension

Statistical regression analysis was used to determine the contribution of each control variable in explaning the observed variance in graph comprehension, as measured by the respondents ability to indicate quantitative results from the visualized data. The biggest contributor to comprehension was the format of the visualization, with respondents performing better when viewing the bubble chart. The next largest contributor was the interactivity of the

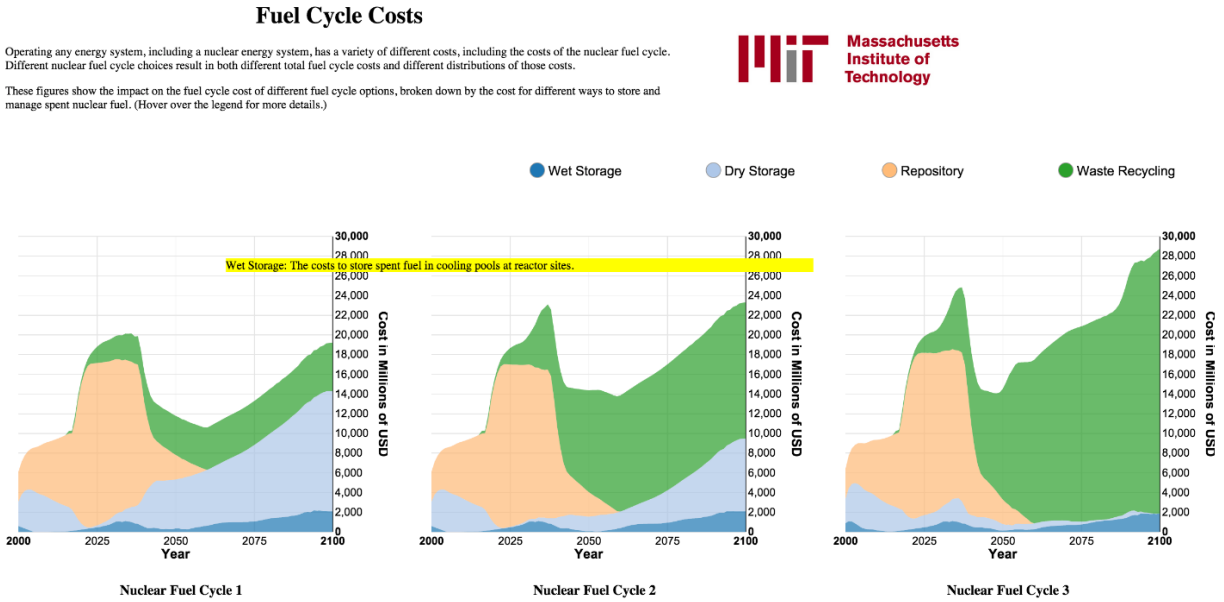


Figure 3.4: Static area chart depicting fuel cycle costs for three different options over time. The interactive version allowed the viewer to query the values at any year.

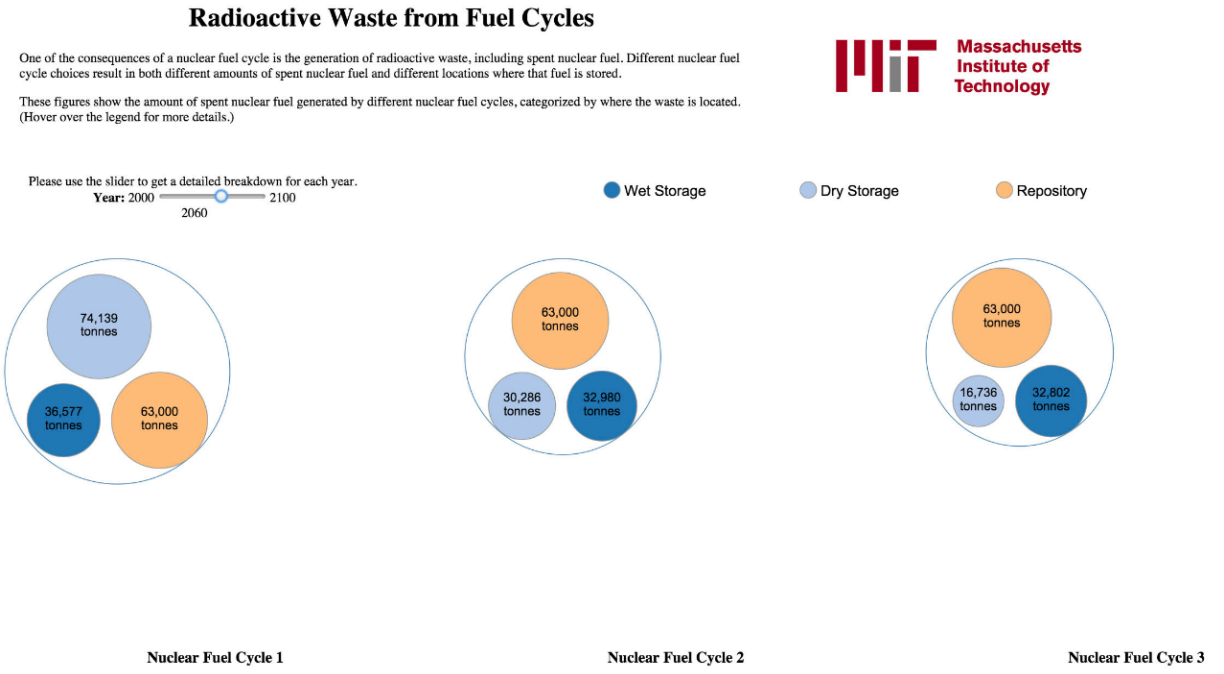


Figure 3.5: Interactive bubble chart depicting the mass of waste for three different options over time. The slider in this version allowed the viewer to see the mass of waste at different years. The static version showed only three specific years for each case.

visualization, with respondents performing better when they viewed an interactive graph, although only for the area charts. While general graph literacy was positively correlated to comprehension, specific domain knowledge did not have an impact.

3.2.2 Decision Quality

Another regression analysis was used to determine the quality of decisions made by respondents, as measured by their ability to identify the best option from among three fuel cycles. Despite the indications that respondents had a better quantitative understanding of bubble charts over area charts, those respondents who viewed area charts were better able to select the option that correpsonded to the requested optimum. Numeracy skills and graph literacy also resulted in better likelihood of success in identifying an optimum.

3.2.3 Credibility

The final analysis considered the perceived credibility of the data shown in the visualizations[7, 8]. The perceived credibility was not impacted by the format of the chart, although respondents did perceive the static charts are slightly more credible than the interactive charts. The visualizations attributed to an academic source were perceived as more credible overall. This was mediated by the respondents' predisposition to trust academic sources relative to governmental sources. Those respondents who reported a higher trust for academic sources perceived more credibility in the data attributed to an academic source, while those respondents who reported a higher trust for governmental sources did *not* perceive more credibility in the data attributed to a government source. Overall, younger respondents were more skeptical of the data. Regarding the visualization itself, the respondents' evaluation of the quality of the visualization, and their own ability to read and use graphical tools in general, were more important for supporting credibility than their measured comprehension of the data.

3.3 Summary

A rigorous social science study of the issues surrounding nuclear fuel cycles was carried out to understand the role of a NFCS in decision making. While a number of stakeholder groups were identified, the bulk of the study focused on federal decision makers and think tank experts. A combination of document review and a dozen interviews were used to identify a set of issues that formed the basis of a survey instrument that was sent to 517 individuals, receiving 137 responses. A principal outcome of the survey was to identify which issues that had been raised earlier were considered important for a NFCS, and separately, which issues the respondents believed could be credibly produced by a NFCS. Two metrics that represented the important issues were used to create a visualization experiment in order to study how different aspects of the visualization impacted the comprehension of the data, the quality of decision making based on that data, and the credibility of the information. While this last step did not use Cyclus visualizations directly, it did provide information about the future direction of visualization of NFCS results.

Chapter 4

Thrust 2: User Interface and Model Generation

With the flexibility afforded by the Cyclus ecosystem, it is important to provide tools that allow users to take advantage of this flexibility while describing a novel fuel cycle. Without *a priori* knowledge of the archetypes that will be used in a simulation, there are three main challenges: (a) ensuring that the fuel cycle represents a physically valid arrangement of facilities, (b) providing a graphical user interface for arbitrary sets of archetype parameters, and (c) allowing for different levels of detail as new archetypes are introduced to avoid overwhelming the user.

A fundamental requirement of this interface is that it presents the design of a nuclear fuel cycle as a user task. Prior NFCSs require the intervention of a developer to introduce novel flows of material between facilities and users are left only with choices of parameters that may distribute material over those flows. In the Cyclus ecosystem, the ad-hoc introduction of new archetypes can enable new flow paths through the system that should be easily realizable as user input. At the same time, a mechanism must exist to prevent or discourage a user from forming fuel cycle flow paths that are not physically realistic. The existence of the DRE mitigates some of the concern for unrealistic flow paths, while a drag-and-drop interface can be implemented to enable the flexible design of fuel cycles.

Thrust 2, led by Dr. Erich Schneider at the University of Texas - Austin, tackled these questions in the development of a user interface for building simulation models. Working closely with the Thrust 4 team, the Java platform was chosen for this capability. A number of design decisions were made to facilitate user interaction, and testing of this interface was carried out. The user input tool was originally referred to as Cycic, but was ultimately incorporated into the broader user interface that also handled output visualization, known as Cyclist[9]. Throughout this chapter, the graphical user capability for building analysis models will be referred to as Cycic.

4.1 Platform Choice

Some consideration was given to the use of Javascript for this functionality and its implementation within a standard web browser. Early prototyping was promising and suggested that

it would be possible to support the core functionality being considered for this tool. However, a desire for integration with the data visualization capability ultimately drove the decision to switch to the Java programming language, based on considerations discussed in Chapter 6.

Java provides native support for many of the user interactions necessary to develop a tool such as this, including file input/output, a standard array of user interface widgets, drag-and-drop capability, and dynamic arrangement of visual elements within the window.

4.2 Design and Layout

The bulk of the activity in this thrust focused on the design of different user interface elements to provide the necessary flexibility, operability and robustness. While the visible layout is important, the most important contributions of this thrust are particular capabilities that are embedded into that layout to support the user experience. Each of the following sections highlights one of those features.

4.2.1 Archetype Discovery

Since each user is free to collect and deploy plugin modules independently, it is necessary for the user interface to have a mechanism for discovering which archetypes are available for any given simulation. This capability is related to the Thrust 0 work highlighted in section 2.2.5. While a user is free to place their archetypes anywhere in their filesystem and refer to them through the fully formed specification, Cyclus is also able to catalog all the available archetypes in a standard location, and report their full specification. When Cycic is started, it may not be aware of any archetypes at all, but can invoke the Cyclus command-line tool to discover which archetypes are available.

Cycic is also able to perform this function using remote execution servers as will be described in Chapter 7. In this case it connects to the remote server and invokes its copy of Cyclus to query which archetypes are available on that server. This allows users to easily switch to different execution servers, including their local computer or any other publicly accessible remote server, and determine what kind of archetypes are available.

Once the list of archetypes is discovered, the user interface itself is updated. A drop-down list of all available archetypes is populated with the information that is returned from these queries, forming a catalog of possible archetypes that a user can choose to include in their model.

4.2.2 Drag & Drop

To use a particular archetype, the user first selects it from the catalog of discovered/available archetypes, placing an icon for that archetype in a "corral" that is then available for inclusion in the model. Simply selecting an archetype is only the first step since the same archetype can be configured in multiple ways to produce different prototypes, each with the same underlying behavior but different parameters. A fuel cycle design is populated with prototypes. For each prototype, the appropriate archetype is added by dragging and dropping its icon from

the corral into the fuel cycle design canvas. Once dropped, these prototypes can be moved around by dragging and dropping to improve the visual layout of the developing fuel cycle.

To configure a prototype, a double-click will open a window that contains widgets for the input of each parameter that must be defined for that archetype, including a name that allows for that prototype to be distinguished from other prototypes of the same archetype. Since Cycic has no prior knowledge of archetype definitions, this must be generated automatically, as described in the next section.

The default representation of archetypes and prototypes is a colored circle, in which the color is automatically determined from the archetype specification. This leads to unique colors for each archetype that permits easy identification of the archetypes being used at any time. It is also possible to define sets of icons that will be used to represent different archetypes and prototypes as they are assigned to specific niches in the fuel cycle.

4.2.3 Automated Input Generation

An important feature related to the automated discovery of archetypes is the ability to automatically generate graphical user interface capability for each archetype. This functionality relies heavily on the metadata discussed in section 2.2.3. Specific metadata annotations have been defined for use by the user interface to enable this. Some of these annotations allow the archetype developer to specify strings that will be displayed as part of the user interface as documentation(e.g. `alias, uilabel, units, doc, tooltip`), while others provide semantic information:

- `type` describes the C++ implementation of this data type and can be used to arrange input widgets in the window for complex data types. For example, if the input quantity is a `std::map< int key, double value>`, the input can be represented as a lists of pairs of input boxes, in which each pair represents a single key/value pair, and button to add more pairs as necessary.
- `uitype` provides a hint about how this variable should be displayed for user input. Some of these indicate that user input for this variable should be a drop-down list populated by other elements of the model: `incommodity`, `outcommodity`, `facility`, `prototype`, *etc.* `combobox` specifies a more generic drop-down list that will be populated by items specified in the `categorical` metadata, while `range` specifies that this input element should be selected from a continuous variable between the values specified in the `range` metadata.
- `range` provides the valid range for an input quantity, allowing archetype developers to ensure that only valid quantities are given.
- `categorical` provides the list of discrete quantities that are valid when the `uitype` is specified as `combobox`, allowing archetype developers to constrain the possible choices of a variable to those that are consistent with the models implemented in the archetype.
- `default` provides a default quantity for the input data, allowing the input widget for this variable to already be populated with that value. The user can still choose to change the value, but immediately sees the default.
- `internal` indicates that this variable should not appear in the user input or Cycic interface. This is reserved for quantities that are otherwise stored in the database, but are initialized indirectly from other user input quantities or default values.

Cycic implements code that makes specific design choices based on the values specified in this metadata. It is able to query this metadata using the Cyclus command-line, whether the archetypes reside on the local computer or on a remote execution server.

Expert Level

One additional metadata annotation is used to provide a specific target capability: varying levels of complexity for users with varying sophistication. The `userlevel` data is an integer from 0 to 10. The user can select their user level, from 0 (simplest) to 10 (most complex), and will only see input widgets that correspond to variables with that user level or lower. All variables with a user level greater than 0 must have a default value. This feature allows archetype developers to encode reasonable defaults for some variables and only allows users to change those values if they have made a conscious decision to access a higher user level.

At this time, the overall design and layout of Cycic does not change with the user level, but this would also be possible. This would allow a completely different look and feel for different user levels in addition to, and possibly informed by, the different set of input data that users are asked to provide.

4.2.4 Commodity Connections

Both conceptually and graphically, prototypes are connected in a nuclear fuel cycle via the commodities that they share. By definition, a commodity does not have any particular quality/composition in Cyclus, but serves to define the markets in which each agent will participate. A user gets to decide which prototypes will serve as consumers and/or suppliers for each commodity in the problem. Cyclus uses the notion of commodity to partition the DRE into smaller network flow problems that can be solved independently and more efficiently. These commodities also result in visual information in the form of arrows that connect prototypes in the fuel cycle design canvas.

In an effort to ensure that the user designs a viable fuel cycle, this information also provides hints to the users about the allowable flows of material among the prototypes that they are placing into the model. It is important to recognize that these are only hints as it doesn't prevent the user from designing a fuel cycle that violates physics. It is, however, up to the agency defined in each facility's archetype to manage its interactions with the DRE, and prevent unphysical results from arising.

4.2.5 Sample

Figure 4.1 shows a partially defined fuel cycle that includes a mine, an enrichment facility and a reactor, using a set of icons based on USDOE graphics for fuel cycles[23]. In this case, the available archetypes have been discovered on the remote Cyclus execution server at the address `http://cycrun.fuelcycle.org`. The uranium mine, enrichment facility and reactor are based on the `Source`, `Enrichment`, `Reactor` archetypes, respectively, provided by the Cycamore library. At the right is the auto-generated user input panel, at user level 0, for the reactor prototype with the name "ALWR". The commodity "U-ore" is produced by the mine and consumed by the enrichment facility while "Fresh-UOX-Fuel" is produced

by the enrichment facility and consumed by the reactor. Note that this latter commodity does not have a specific composition, but that the reactor facility defines a specific recipe, "Fresh-UOX-Fuel-4" that it will request to satisfy this commodity need.

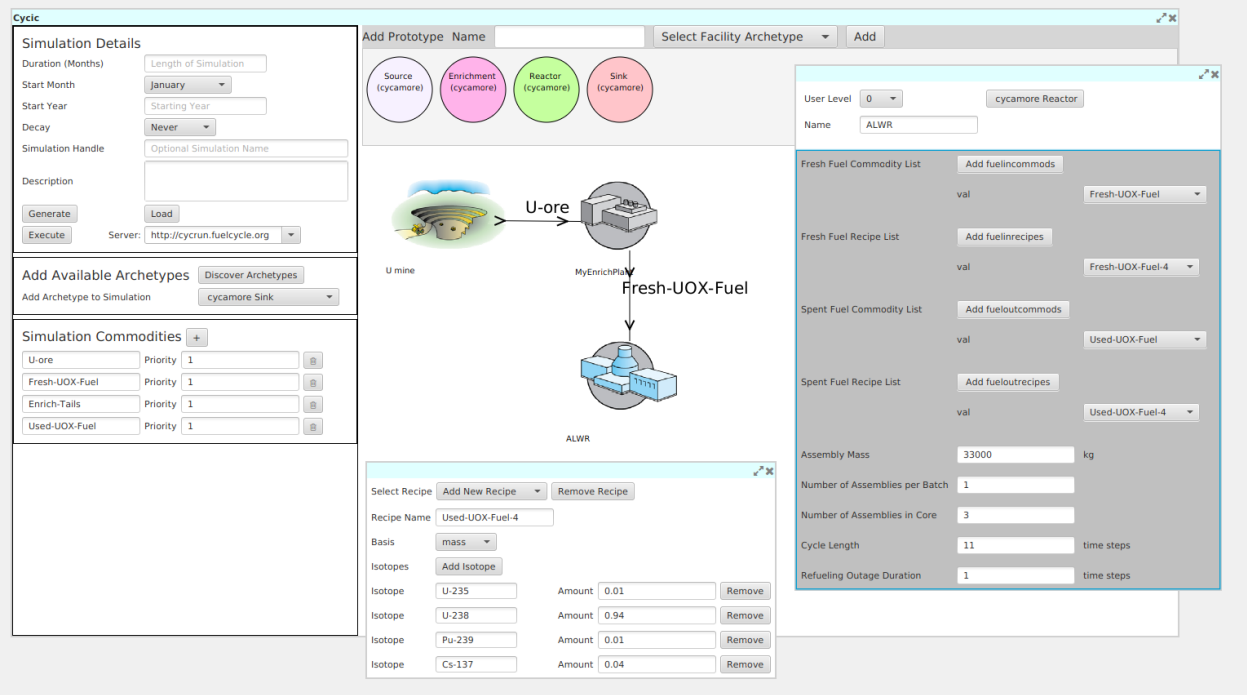


Figure 4.1: A screenshot of the Cycic user interface in the middle of defining a simple fuel cycle.

4.3 User Experience Testing

The Thrust 2 team embarked on user experience testing to determine how well the implemented design of Cycic met the needs of users working to design nuclear fuel cycles. More details of this work can be found in Ref. [10].

The first round of testing relied on a group of graduate students in nuclear engineering. The users were given a tutorial consisting of a brief introduction to the concepts of Cyclus while demonstrating how they were represented in the Cycic user interface. Most users viewed this tutorial once as a group and again individually immediately prior to testing their user experience.

Users were given a task to complete using the software and their actions were tracked by an observer while they also narrated their actions and decisions. The technique of self-narration allows users to comment on things that might not arise in questioning, and also allows the user to complete the task without the interruption of questions. The primary data recorded by the observer was the amount of time and number of mouse clicks required by the user in each step of the task.

Each participant was to construct a simple fuel cycle consisting of three prototypes - a source, a reactor and a sink - and add other components necessary for a full Cyclus

simulation: an institution and a region. Following the task each participant was asked four questions and invited to make additional comments. These four questions focused on specific contrasts that were introduced during the task phase:

- text-based prototype creation vs drag-and-drop
- corral view of regions vs text form of institutions
- the benefit of choosing a low user level
- the clarity of presenting defaults for advanced user level data

The two principal findings of the observations and self-narration were that (a) users were confused about the difference/relationship between institutions and regions, and (b) the text-based institution view was more challenging to understand than the corral-based view of the region interface. While the first of these is a core Cyclus concept and not specific to the user interface, the user interface could be modified to explicitly support a better understanding of the RIF hierarchty. The second finding suggests that the manipulation of institutions should be modified to be more like the region corral view. Two additional, more minor findings were that new prototypes should have a default name, rather than no name at all, and that a better mode for connecting facilities via commodities would be valuable.

Regarding the specific follow up questions, the majority of users (10/13) preferred the drag-and-drop method for adding prototypes, and some of those who indicated a preference for text entry acknowledged that drag-and-drop may be preferred for learning a new tool. Nearly all of the users (12/13) preferred the corral view for regions over the text form for institutions. All users found the presence of user levels made the tool easier to use, reducing clutter and preventing information overload. All users felt that it was clear that values that populated the input widgets when the user level was increased constituted default values.

The software was updated prior to a second round of testing. Specifically, the institution view was updated to behave similarly to the region view, both now relying on an updated corral view with drag and drop capability from available archetypes. In addition, the region view now also includes a list of unassociated institutions to reinforce the relationship between regions and institutions. Tooltips (small documentation that appears when the mouse hovers over an input widget) were added to the region and institution views, and popup help windows were implemented with longer documentation for most windows and widgets.

A similar tutorial was given prior to the user experience testing in this second round. The task again required users to build a fuel cycle model, but this time with more complexity. The facility prototypes consisted of three sources, a fuel fabrication plant, a reprocessing plant, two different reactors, and sink. The institution was configured to build reactors of each kind at three different points in time. This more complex scenario was constructed to provide more time for observation and self-narration. In addition observing the time spent and number of clicks in each view, the observer also recorded the number of times that the user switched between two open views.

Fifteen users were again asked a series of questions following the exercise. For those users who had participated in the previous test, they were asked to rate the changes. Those users new to Cycic were asked to rate the same features as in the first test, as well as the new features represented by the software changes. All participants were asked how each view helped to understand Cyclus itself and to rate different widget choices. Finally, participants were solicited for any improvements that they wold recommend.

The average time to complete the task was about 20 minutes (range: 16-30 minutes),

nearly 2 times longer than an expert user as represented by the software developer. The majority of this time (12 min) was spent in the configuring the facilities in their respective form views, but the range of times was narrowest, with the slowest user only taking 72% longer than the fastest. The recipe view provided the most challenge, in this respect, with the slowest user spending nearly 6 times longer than the fastest person.

The most significant difference between the expert time to complete and the users was with the institution form. While the ratio of 3.7 was similar to that in the region view (3.8), the time spent in the institution view represented a substantial larger fraction of the total (3.5 min vs 1 min). Through questioning, the users indicated that the specific purpose of this institution archetype was unclear and thus the requested data was also unclear.

Combining the observed interactions with the responses to the post-exercise questions, much of the feedback of the second round of testing was consistent with the first round. Returning users recognized the improvements in the institution view but still considered one of the least clear parts of the problem definition. The addition of tooltips was not well received since the value of information was deemed too low considering the time delay for the tooltip to appear.

4.4 Summary

A number of design choices were implemented in the Cycic model building interface to satisfy the need for supporting a flexible plugin ecosystem. These tools have been demonstrated to support the construction of complex fuel cycle models. Testing of the user experience resulted in software improvements that have been favorably received.

Chapter 5

Thrust 3: Metric Translation

The standard output of a NFCSs, in general, consists of material inventories and flows, each with a composition that may change over time, and possibly facility deployment histories including time varying capacity factors. Most users of these tools, however, are interested in quantities that are derived from these fundamental data. In the simplest case, such as decay heat and radiotoxicity, the metrics of interest may be easily derived from the material inventories and flows using tabulated data. In more complicated cases, however, it may be necessary to perform more complex post-processing. Many socioeconomic metrics, including economic analysis, require this kind of approach.

Thrust 3 was led by Paul Wilson at the University of Wisconsin-Madison, to provide a mechanism to translate the fundamental data stored by Cyclus into metrics of interest to a broad set of users. This thrust was impacted by the change in scope to include Thrust 0, and only a basic demonstration of this capability was accomplished.

Since the output data from Cyclus is even more obscure than the material inventories and flows common in other tools, the first step in this thrust was to make that conversion. With that information available, an extensible tool was developed that allowed users and analysts to introduced new complex metrics by defining their dependency on simpler metrics. The Cymetric tool, described below, supported the automatic resolution of metric dependency all the way to the fundamental simulation data. This thrust also informed a handful of metric translations for inclusion in Cyclist as described in Chapter 6. Finally, some exploratory work for economic analysis of transitions was completed using the Cymetric tool.

5.1 Fundamental Data

The fundamental output of Cyclus is optimized to correspond to the discrete data and discrete facility paradigm: a list of material transactions as a function of time in which the composition of each material object is stored separately in order to reuse compositions where appropriate. While this format leads to more compact output databases from Cyclus simulations, it is not a natural way to analyze fuel cycle performance.

Converting a stream of transactions to inventories and flows is relatively straightforward. Conceptually, each transaction that involves a given facility represents a material addition or subtraction to the inventory at that facility. A separate database query is necessary to access

to composition of that transaction, or possibly multiple compositions if multiple discrete objects were traded. Combining the quantity and composition of each object involved in the transaction allows the overall composition of the inventory to be updated. For some facilities, material objects are created and/or destroyed according to the physics models that they represent. These actions are recorded in separate database tables that must also be tallied to get an accurate accounting of the inventories in any facility at any point in the simulation. In summary, each material object is tracked from its creation to its destruction, adding or subtracting it from the inventory of the appropriate facility as each operation and transaction occurs.

This capability was implemented in a stand-alone tool known as Cyan[11]. In addition to deriving the time-dependent inventories in each facility, it tallied cumulative flows among facilities and can produce graphical representations of the fuel cycle indicating the flows of material along the commodity arcs between facilities.

After experience with Cyclus on a number of problems, it became clear that the complexity of performing this operation robustly during post-processing was overly burdensome compared to the cost of extending the fundamental Cyclus database to contain this information. An option was added to the Cyclus command-line tool to automatically accumulate the inventories in each facility as a function of the simulation time.

5.2 Cymetric

In the spirit of the extensible nature of the Cyclus ecosystem, the post-processing tools were also designed to be extensible. This extensibility is valuable for possible future metrics that are derived from the generic data stored in a Cyclus output database, as well as for metrics that might be based on custom tables that are associated with a particular archetype. The goal of this thrust does not include the visual representation of metrics, so focuses on generating new database tables with metrics of interest, derived from the original database tables. Python was chosen for this tool, known as Cymetric, because of the flexibility it would allow for using its capabilities: as a command-line tool for performing small operations on a database, as a module to be included in scripts for batch processing of databases, or using Jupyter notebooks for more interactive exploration of data. These different modes imply different levels of user sophistication. The interactive mode is best employed by sophisticated Python users, but could result in packaged scripts that could be used by less sophisticated users to repeatably generate visualizations of interesting results.

With similar concepts to the automatic code generation discussed in section 2.2.1, Cymetric provides streamlined capability for defining new metrics based on previously defined metrics using the Pandas data analysis framework. The fundamental data structure, known as a dataframe, is a table in which each row represents a record of related data. Pandas provides compact mechanisms to perform complex operations on such data tables.

To define a new derived metric in Cymetric, a user/developer must provide the dependencies, the format of the output dataframe, and the algorithm for populating the output dataframe from the dependencies. For each dataframe upon which this metric depends, it is also necessary to define the columns to use as the index and the columns to use as the value. For the output dataframe, it is necessary to define which columns it will contain, formally

known as a schema. In many instances the algorithm that calculates the output dataframe from the input can be compact and developers are encouraged to design metrics in this way, possibly chaining multiple metrics to achieve an ultimate calculation. Such a design approach allows the intermediate metrics to be available as dependencies for other metrics developed in the future.

Listing 5.1 shows a listing of a simple metric that defines the activity of material objects as a function of the mass and composition of that object. This new metric depends on the `Materials` table, in which the index is formed from a number of simulation-related identifiers as well as the time of creation and the nuclide id, and the value is the mass of the nuclide. The resultant activity is stored in a table with columns defined from the index of the dependency, and adding a column for the activity. The algorithm simply loops through the rows of the mass dataframe, uses the decay constant and atomic mass to determine the total activity of that nuclide, and appends it to the list of activities. It is straightforward to imagine a derived metric for decay heat that depends, in turn, on this activity metric.

Listing 5.1: A sample derived metric in Cymetric

```
# Activity (mass * decay_const / atomic_mass)
_actdeps = [
    ("Materials", ("SimId", "QualId", "ResourceId", "ObjId", "TimeCreated", "NucId"),
        "Mass")
    ]

_actschema = [
    ("SimId", ts.UUID), ("QualId", ts.INT),
    ("ResourceId", ts.INT), ("ObjId", ts.INT),
    ("TimeCreated", ts.INT), ("NucId", ts.INT),
    ("Activity", ts.DOUBLE)
    ]

@metric(name="Activity", depends=_actdeps, schema=_actschema)
def activity(series):
    """Activity metric returns the instantaneous activity of a nuclide
    in a material (material mass * decay constant / atomic mass)
    indexed by the SimId, QualId, ResourceId, ObjId, TimeCreated, and NucId.
    """
    tools.raise_no_pyne("Activity could not be computed", HAVE_PYNE)
    mass = series[0]
    act = []
    for (simid, qual, res, obj, time, nuc), m in mass.iteritems():
        val = (1000 * data.N_A * m * data.decay_const(nuc) \
              / data.atomic_mass(nuc))
        act.append(val)
    act = pd.Series(act, index=mass.index)
    act.name = "Activity"
    rtn = act.reset_index()
    return rtn
```

5.3 Demonstration with Economic Analysis

A set of derived metrics were created to demonstrate how Cymetric could be used to estimate economic performance of different fuel cycles. A simple economic model for a reactor was introduced that used data from the Cyclus output data base to calculate various estimates

of economic performance. The economic model included components for the construction costs, the fuel cycle cost expressed as a variable cost, operations and maintenance costs including both fixed and variable costs, and decommissioning costs. This model used the following data for each reactor from the Cyclus output database:

- the date of commissioning,
- the lifetime,
- the reactor power,
- the energy generated as a function of time, and
- the mass of fuel transacted as a function of time.

The construction costs were determined by combining the power level of the reactor with user-provided overnight capital cost, construction time and payment schedule. The payment schedule spreads out the capital cost over a period that is greater than the construction time to convert from an overnight capital cost to a more realistic construction cost. The operations and maintenance costs were determined by combining the power level of the reactor with a user-provided fixed cost and the total power generated with a user-provided variable cost. Fuel costs are a product of the transacted mass of fuel with a user-provided fuel cycle cost. Finally, the decommissioning costs use a model similar to the construction cost model. While a simple model, it is able to demonstrate the generation of a time-dependent cash flow based on the time-varying Cyclus output data and can also easily incorporate stochastic realizations of the individual costs.

Because the output of this model is a series of cash flows, it is possible to examine a variety of approaches to translate the cash flows into simpler metrics for comparison. The most common approach is to calculate a levelized cost of electricity (LCOE) by summing the present value of the lifetime costs and dividing by the present value of the lifetime energy generation, assuming some discount rate. This approach is useful for evaluating a single reactor, but becomes less useful when evaluating a strategy in which multiple reactors are built, possibly with different costs and over long periods of time. One variation relies on first calculating the LCOE for each reactor, and then determining a time varying fleet-wide LCOE by averaging the LCOE of all operating reactors at each time step, where the average is weighted by the power level of the reactors. Another variation uses a formulation similar to the LCOE but instead of performing the sum over the entire lifetime, it is performed over a smaller time window, and the costs and generation are summed over all operating reactors at each time step. The value reported at each time step is based on a calculation window that starts in that time step.

This capability was used to look at the economic performance of a transition from thermal to fast reactors consistent with the EG23 transition identified by the department of energy. Figure 5.1 shows three different representations of the time dependent costs, depending on how they are averaged over time and over the complete fleet of reactors. The left-most chart in Figure 5.2 shows the total power generation in this scenario. This is helpful to understand the other two charts that show the net accumulated profit over time, both with discounting to the present value (center) and without (right). Both figures show how typical discounting can render decisions in the distant future irrelevant to the present value of the economic analysis.

More details on the implementation and demonstration of this capability is available in [12].

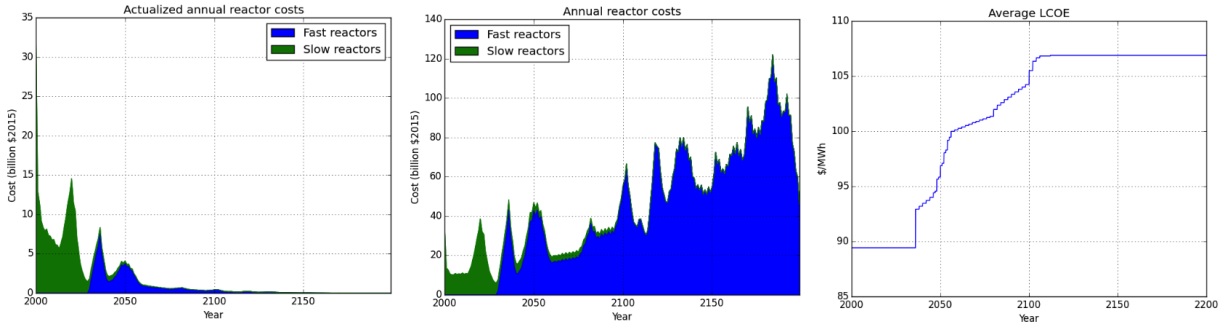


Figure 5.1: Three different representations of the time-dependent costs of an EG23 transition: (left) the present value, (center) the as-spent, (right) averaged LCOE

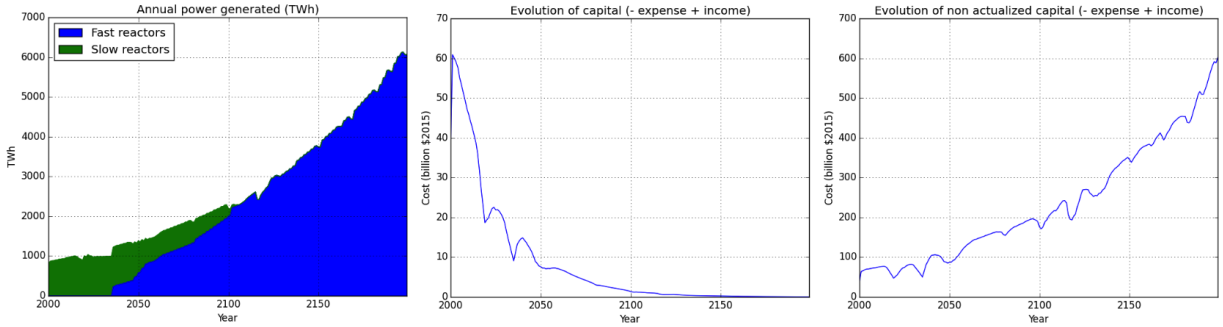


Figure 5.2: The time-dependent energy generation and net profit (with and without discounting) of an EG23 transition.

5.4 Summary

Cymetric offers a demonstration of an extensible system to define metrics for nuclear fuel cycle analysis based on the output database of a Cyclus simulation. As in thrust 0, automated code generation reduces the burden on individual developers to understand either the underlying data access methods or the dependency resolution of the extended metrics.

Chapter 6

Thrust 4: Visualization Environment

The fidelity and volume of data that can be provided by a discrete facility, discrete material object agent-based fuel cycle simulator opens the door for a richer visualization experience. The fundamental transaction data, as described in Chapter 5, can be considered most abstractly as the mass of each transaction sparsely mapped onto a five-dimensional domain: time, sending facility, receiving facility, material ID, nuclide. The simplest visualizations are formed from filtering this data set on some domains and aggregating it on other domains. For example, the total time-dependent flow between two facilities is found by filtering the sending/receiving facility domains with those two facilities, and aggregating over the material id and nuclide domains. Being able to easily select the filters and aggregations can enable data exploration that may not be natural through more typical spreadsheet based data analysis environments. Additional power can be derived from examining different linked visualizations in which the filter and aggregation operations on one are reflected on the other.

Thrust 4 was led by Yarden Livnat at the University of Utah, to develop a complete visualization environment that can enable a richer data exploration experience. The lack of a robust NGFCS in the early stages of the project limited the scope of this thrust. While this thrust originally aimed to explore different visualization modes for different levels of nuclear fuel cycle expertise, it focused instead on providing visualization capability for Cyclus for developers and expert users. This was driven, in part, by a need for Cyclus developers to visualize results during the development process. This thrust was also somewhat hampered by a lack of real-world analysis needs within the project. Without interesting output analysis, it was difficult to motivate advanced visualization needs despite the availability of a richer data set.

In order to simplify distribution across many types of computing systems, and to take advantage of existing high quality visualization toolsets and user interface widgets, the JavaFX software platform was selected for this thrust. This decision ultimately drove the software platform decision of Thrust 2, as well. JavaFX is a freely available platform that is supported on a wide variety of computing systems.

6.1 Cyclist

The tool for data visualization is named Cyclist, and ultimately incorporated the Cycic model building environment as a single platform-independent tool. It is a stand-alone application that can read Cyclus databases from the local computer or access a remote execution server to download and access a database. Upon loading a database, Cyclist checks for the existence of an inventory table in the database, as described in section 5.1 and reconstructs the inventory if not present using Cyan-based methods. Cyclist then offers a list of database tables to the user from which to select data to visualize. Once a table is selected, each of the columns of that table are offered in a separate list as specific data to include in a visualization. The other columns in that table become available for other operations such as aggregation or filtering.

6.1.1 Workspaces

The visualization environment is arrange in workspaces, each of which may contain multiple individual visualizations. Some characteristics of the workspace apply to all the visualizations in that workspace, allowing for operations to be applied to a workspace such that its effects are reflected in all visualizations simultaneously. Multiple workspaces can be opened simultaneously, each with different settings and operations applied. A workspace can be duplicated, thus allowing for largely the same visualizations, but with slightly different operations or slightly different parameters for those operations.

6.1.2 Drag & Drop

Most operations in a workspace are implemented by a drag-and-drop interaction mode. Once a new chart has been created, the a database table columns can be dragged and dropped to populate the X and Y axes, as well as the "Group by" entry which enables aggregation and filtering (see the next section). Depending on the data types of the columns chosen for the X and Y axes, a set of operations will be offered that can be applied to that data. For example, in some circumstances it may be logical to sum the value in the Y axis for each point on the X axis. The type of plot is automatically adapted to the type of data being plotted. For a continuous variable, e.g. Time, the plot will be a typical x-y scatter/line plot. For a discrete variable, e.g. Prototype, the plot will be a column/bar chart.

6.1.3 Aggregation & Filters

Additional columns can be dragged into the "Group by" entry to create categories for the operation being applied to the Y axis data. For example, if the Y-axis data is being summed over all rows for each X axis point, the addition of a grouping field will separate the single sum into multiple sums, each associated with a different result in the grouping column.

In a similar fashion, additional fields can be dragged into the title bar of the plot to create an opportunity for filtering. For each field in the filtering bar, a new user interface appears that allows the user to select from the values in that field. For fields that represent discrete quantities, the filter interface appears as a set of checkboxes and the user can select

which values are shown or not. For fields that are represented as continuous quantities, the users specifies the maximum and minimum, either with a slider or with text boxes depending on the required precision. It is possible, but not required, for fields to be used for both aggregation and filtering, increasing the flexibility of generating plots.

Figure 6.1 shows a sample plot with aggregation across multiple fields and filtering across multiple fields. In this case, the total quantity of material in inventory as a function of time. Since there may be many discrete inventories, the Y-axis value is summed over all possible inventories. Without grouping, this would result a single curve on the plot. However, in this problem, the quantities are grouped by both Prototype (of which there are three) and Nuclide (of which there are four). Therefore, there is a single curve for each combination of Prototype and Nuclide. This plot also includes filtering by Prototype, Nuclide and Time. The user has selected to see all Prototypes, but only the fissile nuclides. Finally, the user has asked to see only the first 60 time steps. Since Time also defines the X-axis, this serves to zoom in the plot to those 60 time steps.

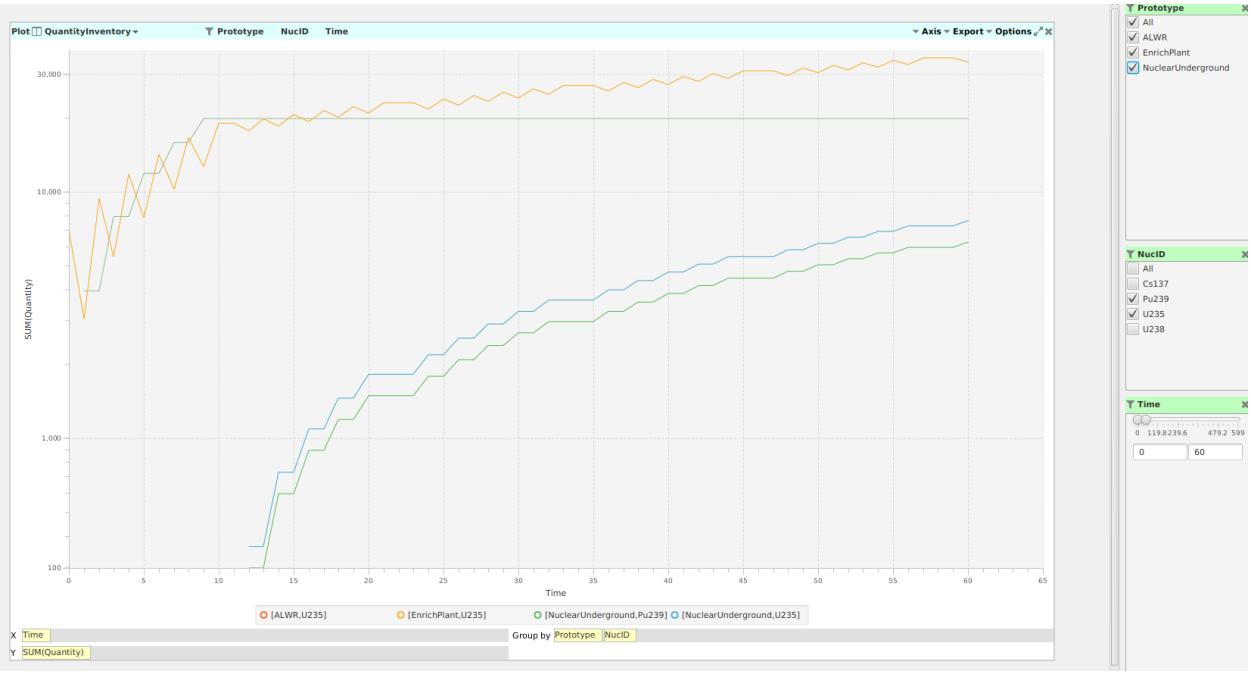


Figure 6.1: A typical plot showing the quantity of material inventory at each point in time. This case shows the application of a SUM operation to the Y-axis data, grouping of the results by Prototype and Nuclide, and subsequent filtering by Prototype, Nuclide and Time. Since Time is a continuous variable, the user interface for this filter is different. Since Time is the X-axis variable, its filter is applied differently.

6.1.4 Extensibility

Since it is possible for many tables to exist in the Cyclus output database, Cyclist shows only a default set of tables in its interface. There is also a mechanism, however, to add any table, including custom tables, to that list. Thus, if an archetype developer adds a custom

table to the database it is still possible to use Cyclist to visualize the data it contains with no changes to Cyclist.

6.2 Custom Visualization: Material Flows

Although the primary visualization modes in Cyclist are based on well-understood x-y plots, there was also a desire to implement application specific visualizations to support fuel cycle analysis. Following the network flow model that underlies the Cyclus, a material flow visualization was created to demonstrate an interactive and exploratory visualization mode.

As shown in Figure 6.2, this visualization provides a way to examine both total flows and time dependent flows between sets of agents. Both the sender and receiver can be used to specify archetypes, prototypes, or individual agents, whether regions, institutions or facilities. Multiple pairs of flows can be included for comparison purposes. As each pair is added to the flow map, its cumulative flow is added to the strip chart below. The checkboxes on the let allow for filtering by the commodity and by the nuclide. Finally, a time window can be specified over which to sum the flow as reported on the flow diagram.

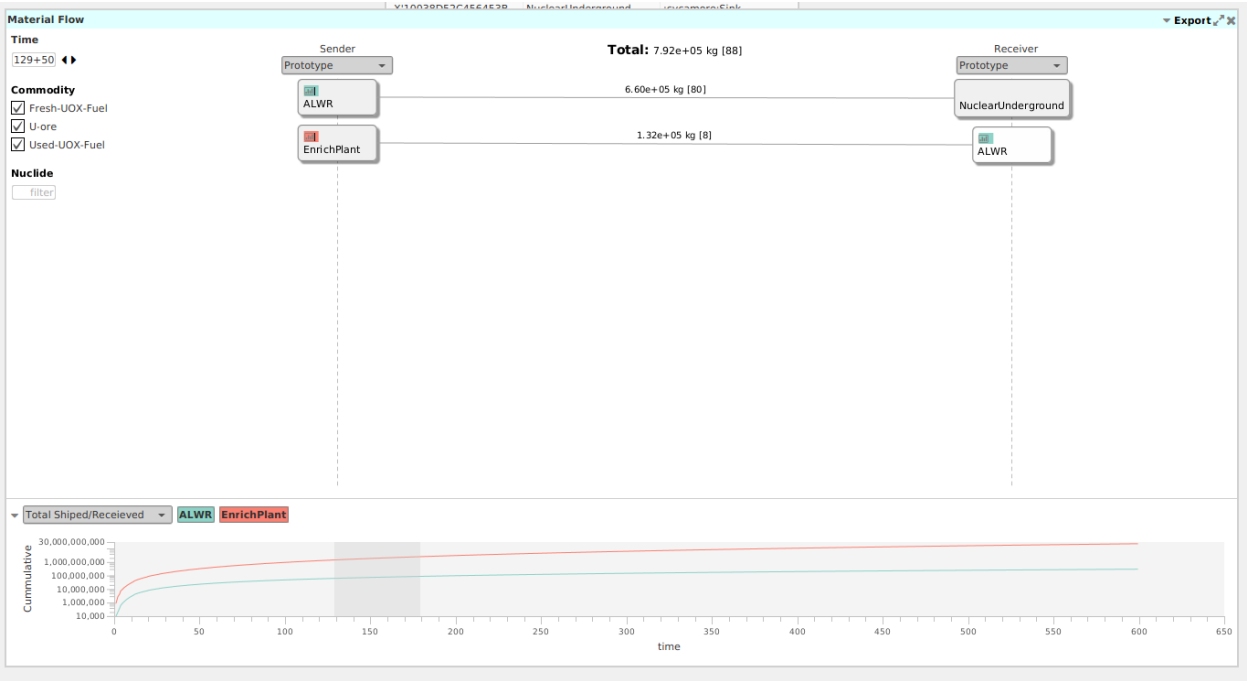


Figure 6.2: A flow visualization shows the total flow between different pairs of prototypes, aggregating over all agents that share that prototype. The upper pane shows the sender receiver pairs while the lower pane shows the cumulative flow as a function of time. The quantity shown in the upper pane is the total flow contained in the window indicated in gray from time step 129-179.

Different visual representations of this flow information have been conceptualized for future development, including a map-based view in the event of trade among many regions, or a graph-based view for representing flows throughout the whole fuel cycle.

6.3 Limitations

There were some limitations identified during the development of Cyclist. First and foremost, as Cyclus simulations became more complicated, the volume of data began to become an obstacle. This was true both in terms of accessing the database file as well as managing the data in memory. This is particularly true for the operation that builds an inventory table from the transaction tables. Furthermore, there is growing interest in processing results for 100's or 1000's of separate but related Cyclus runs either due to parametric sweeps, sensitivity studies, or optimization capability. The current Cyclist infrastructure is not well suited for such applications.

In addition, since the beginning of this project, large investments have been made by various third parties in the technology for Javascript-based software. Visualization solutions and user interface widgets in Javascript have achieved a quality that surpasses the JavaFX version, propelled by rapid development of web apps for a variety of computing platforms, particularly mobile. A new project has begun that is expected to replace Cyclist with a Javascript based interface and rely more heavily on intermediate computing capability to process results before reaching the device that will actually display the results.

6.4 Summary

A dedicated, open source visualization tool was developed to facilitate the display of Cyclus databases in creative ways. This tool was designed with an advanced user in mind - one who could interpret the names of fields and tables in the output data base. The basic framework for this visualization tool could be used for more creative visual representations and to craft simplified tools for different categories of users. The Cycic model building tool was integrated into Cyclist so that users had a uniform experience in both generating their model and analyzing the results. Both tools support remote execution (see Chapter 7) and therefore Cyclist became a convenient way to distribute access to Cyclus without requiring users to install it independently.

Chapter 7

Thrust 5: Efficient Design of a Client-Server Model

One goal for a NGFCS is deployment on a variety of computing platforms, including those with greatly reduced resources (mobile/tablet). This suggets a client-server approach in which the simulation itself occurs on a remote execution server and only the visualization occurs on a local device. To ensure that the user experience meets expectations on the local device, the software must be designed to accommodate, as much as possible, the limitations of the network connection. Another advantage of a client-server strategy is the ability to scale up the computing resource on the server side, as necessary for larger and more complex problems.

Thrust 5 was led by Robert Hiromoto at the University of Idaho, to understand the network requirements and limitations of deploying a client-server computational model. Like some of the other thrusts, this task was undermined, in part, by the lack of a fully functioning NGFCS. Since this thrust also depended on a demonstration of the visualization software that would run locally, that was itself delayed by the lack of a NGFCS at the projects start, it was largely reformulated.

The outcome of Thrust 5 was a lightweight server that would listen for Cyclus tasks, perform the simulation, and offer the output database for further analysis. This capability was deployed in two related capacities: as a remote execution server for Cyclus users and as a compute farm for large scale analysis. This became a key component to providing platform-independent access to Cyclus when coupled with the Cyclist tool for model building and data visualization.

Cloudlus[13] is a lightweight wrapper, written in the Go programming language, that listens on an internet port for tasks, performs those tasks by launching a Cyclus simulation, and then serves the output database on the same internet port. Cloudlus has been deployed continuously since 2014 as the mechanism for users to submit Cyclus jobs through the website, www.fuelcycle.org. The same Cloudlus service is available to users of Cyclist as a remote execution service as described briefly in Chapters 4 and 6.

Cloudlus also supports a mode for large scale deployment and analysis. In this mode, many Cloudlus workers are deployed in cloud computing resources and register with a single master process that then issues requests to those workers. The master can be designed/configured to generate that work in many different ways. Cloudlus has been deployed on commercial cloud

resources and UW-Madison's high-throughput computing (HTC) computing system to support large scale optimization[24]. In this case, the master implemented an optimization algorithm that relied on 1000's of independent Cyclus simulations, with that master automatically determining the input file for each simulation based on the results of previous simulations.

This infrastructure has been sufficient, to date, but may not scale well as problems become more complex due to the size of the output database. Storage and network transfer of those databases may become a challenge as more complex problems result in databases of many gigabytes. It will continue to be sufficient for beginner use through the website or the Cyclist interface, but as more robust and scalable replacements for Cloudlus emerge, better remote execution paradigms are also expected to merge.

Chapter 8

Summary

This project made substantial progress on its original aim for providing a modern user experience for nuclear fuel cycle analysis while also creating a robust and functional next-generation fuel cycle simulator (NGFCS).

Early changes in project scope required more investment in the generation of a NGFCS that originally planned, delaying progress in some of the thrust areas and ultimately preventing the project from being as tightly integrated as intended. One of the most important aims that was not accomplished was a demonstration of substantially different user experiences for different categories of stakeholders. Most thrusts areas scaled back the number of stakeholder groups to a single group, most often a sophisticated nuclear engineering stakeholder group. While Thrust 1 did provide insight into the interests of a different stakeholder group, namely national-scale decision makers, there was no opportunity to demonstrate a user experience that suited this group.

The success of the Cyclus kernel did catalyze a growing community of contributors to an active ecosystem. A number of institutions have since been funded to contribute an array of alternative archetypes for facilities including enrichment facilities, fuel fabrication facilities and reactors. Additional investments were made in optimization efforts related to Cyclus including the detailed development of the dynamic resource exchange (DRE) and an external optimization wrapper that was able to identify an optimal transition strategy.

Future efforts should focus on further developing post-processing capability across a variety of different stakeholder groups. While there is constant demand for more visualization capabilities for advanced users, there continues to be value in the original premise of providing different visualizations to different audiences, based on the same simulation kernel.

In addition, it will be valuable to see Cyclus applied to real fuel cycle analysis. This kind of application is critical to all aspects of continued progress in the Cyclus ecosystem. Most importantly, it will uncover and help prioritize important needs in the user experience. It will undoubtedly also lead to the development of new archetypes that are customized to suit the particular fuel cycle problem being solved. Through these developments, the community will identify common improvements, needs and directions for the ecosystem as a whole.

Products

[1] K. D. HUFF et al., "Fundamental concepts in the Cyclus nuclear fuel cycle simulation framework," *Advances in Engineering Software*, **94**, 46 (2016).

[2] R. CARLSEN et al., "Cyclus 1.5.0," *Figshare* (2016), http://dx.doi.org/10.6084/m9.figshare.4312643.v2.

[3] R. W. CARLSEN and P. P. WILSON, "Challenging Fuel Cycle Modeling Assumptions: Facility and Time Step Discretization Effects," *Nuclear Technology*, **195** (2016).

[4] R. W. CARLSEN et al., "Cycamore v1.0.0," *Figshare* (2014), http://figshare.com/articles/Cycamore_v1_0_0/1041829.

[5] A. M. Scopatz et al., "Cyclus Archetypes," *ArXiv e-prints* (2015), https://arxiv.org/abs/1511.05619.

[6] N. LI, *The Science-Policy Interface as a Communication Process: Exploring How Policy Decision-Makers Perceive Science-Driven Policy and Make Evidence-Based Decisions on the Nuclear Fuel Cycle*, PhD thesis, University of Wisconsin, Madison, WI, United States, 2015.

[7] N. LI et al., "Visualizing scientific data for lay audiences: Effects of graphical characteristics on comprehension and confidence in data quality," *Proc. Proceedings of the International Communication Association*, San Juan, Puerto Rico, 2015.

[8] N. LI, D. BROSSARD, D. SCHEUFELE, P. WILSON, and K. M. ROSE, "Communicating data: Interactive infographics, scientific data and credibility," *Visual Communication Quarterly* (under review).

[9] Y. LIVNAT et al., "Cyclist: a visual interface companion for the Cyclus project," 2015, https://github.com/cyclus/cyclist2.

[10] R. FLANAGAN, *Novel Methods for Generalizing Nuclear Fuel Cycle Design, and Fuel Burnup Modeling*, PhD thesis, University of Texas, Austin, TX, United States, 2015.

[11] R. W. CARLSEN, "CyAn," *Figshare* (2014), http://dx.doi.org/10.6084/m9.figshare.1041836.

[12] V. CLOîTRE, "Performing nuclear fuel cycle analysis with Cyclus," Master's thesis, École Polytechnique, Université Paris-Saclay, Paris, France, 2015.

[13] R. W. CARLSEN, "Cloudlus," 2014, https://github.com/rwcarlsen/cloudlus.

[14] M. GIDDEN, P. WILSON, K. HUFF, and R. CARLSEN, "Once-Through Benchmarks with C YCLUS, a Modular, Open-Source Fuel Cycle Simulator," *Proc. Proceedings of the 2012 ANS Winter Conference*, San Diego, CA, 2012.

References

[15] A. M. YACOUT, J. J. JACOBSON, G. E. MATTHERN, S. J. PIET, and A. MOISSEYTSEV, "Modeling the Nuclear Fuel Cycle," *Proc. The 23rd International Conference of the System Dynamics Society," Boston*, 2005.

[16] L. VAN DEN DURPEL, A. YACOUT, D. WADE, T. TAIWO, and U. LAUFERTS, "DANESS V4.2: Overview of Capabilities and Developments," *Proc. Proceedings of Global 2009*, Paris, France, 2009.

[17] J. J. JACOBSON et al., "VISION User Guide-VISION (Verifiable Fuel Cycle Simulation) Model," , Idaho National Laboratory (INL) (2009).

[18] L. GUERIN and M. KAZIMI, "Impact of Alternative Nuclear Fuel Cycle Options on Infrastructure and Fuel Requirements, Actinide and Waste Inventories, and Economics," Technical Report MIT-NFC-TR-111, Massachusetts Institute of Technology (2009).

[19] L. GUERIN et al., "A Benchmark Study of Computer Codes for System Analysis of the Nuclear Fuel Cycle," Technical Report, Massachusetts Institute of Technology. Center for Advanced Nuclear Energy Systems. Nuclear Fuel Cycle Program (2009), Electric Power Research Institute.

[20] C. M. MACAL, "To agent-based simulation from system dynamics," *Proc. Simulation Conference (WSC), Proceedings of the 2010 Winter*, pages 371–382, IEEE, 2010.

[21] M. J. GIDDEN, *An Agent-Based Modeling Framework and Application for the Generic Nuclear Fuel Cycle*, PhD thesis, University of Wisconsin, Madison, WI, United States, 2015.

[22] P. P. H. WILSON et al., "Market-Based and System-Wide Fuel Cycle Optimization," (2017).

[23] B. DIXON et al., "Dynamic Systems Analysis Report for Nuclear Fuel Recycle," INL/EXT-08-15201, Idaho National Laboratory (2008).

[24] "Center for High Throughput Computing," http://chtc.cs.wisc.edu.

[25] K. HUFF and B. DIXON, "Next Generation Fuel Cycle Simulator Functions and Requirements Document," fcrd-sysa-2010-000110, Idaho National Laboratory (2010).